JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

Studies on Nonlinear Real Arithmetic Satisfiability

by

Tatuya Kamada

August 2016

Abstract

Nonlinear real arithmetic problem is to find the solutions in systems of equalities or inequalities. The satisfiability problem is to decide if a given constraint in conjunctive normal form admits a satisfiable assignment. In this research report, we study the satisfiability problem of nonlinear real arithmetic.

For nonlinear real arithmetic problem, Collins created Cylindrical Algebraic Decomposition (CAD) in 1975. For the satisfiability problem, SAT solvers and the algorithms have been remarkably improved since mid 1990s.

We study one of the most efficient nonlinear real arithmetic solver: NLSAT. We make sure the strategy for the performance through giving its algorithm and investigating the source code. We also survey the algorithms of cylindrical algebraic decomposition to understand the NLSAT implementation.

Contents

Chapter 1

Introduction

Nonlinear real arithmetic problem is to find the solutions of x_i in systems of equalities or inequalities $\mathcal{P}$

$$\mathcal{P} = \left\{ \begin{array}{lll} P_1(x_1, \ldots, x_i, \ldots, x_n) & \rhd_1 & 0 \\ P_2(x_1, \ldots, x_i, \ldots, x_n) & \rhd_2 & 0 \\ \vdots & & \\ P_j(x_1, \ldots, x_i, \ldots, x_n) & \rhd_j & 0 \end{array} \right\}$$

where $\rhd_j$ is either " $<$ " or " $=$ ", P_j is a n-variant k degrees polynomial such that

$$c_{n_0} x_n^k + c_{n_1} x_n^{k-1} + c_{n_2} x_n^{k-2} + \cdots + c_{n_k}$$

where coefficients c_{n_k} are $x_1, \ldots, x_{n-1}$-variant polynomials with integer coefficients, and $x_1, \ldots, x_n$ are variables over reals.
This class of problem has a broad applications such as Polynomial Optimization, Geometric Modeling, Robot Motion Planning, and Stability Analysis [1]. Moreover, the problem is proved decidable in 1951 by Tarski [2], and in 1975, Collins invented a significantly improved algorithm: "Cylindrical Algebraic Decomposition" for the problem in [3]. However, Davenport and Heintz show the complexity is doubly exponential in [4], meaning that the size and the number of variables are strongly limited.

At the same time, there is a class of satisfiability problem. That is to decide if a given constraint in conjunctive normal form admits a satisfiable assignment. Nonlinear Real Arithmetic Satisfiability is a problem such that

$$\exists x_1, \ldots, x_n (\mathcal{P}).$$

The complexity of satisfiability problem is NP-complete [5], yet, the solvers and the algorithms have been remarkably improved since mid 1990s [6]. For example, modern SAT solvers solves $1,000,000+$ variables boolean satisfiability problems in a few seconds to a few minutes in 2011 [7]. Modern Nonlinear Real Arithmetic satisfiability solvers solves $n = 10+$ variables, $k = 6+$ degrees problems in a few seconds to a few minutes in 2013 [8].

Under those circumstance, we study one of the most efficient nonlinear real arithmetic solver: NLSAT. We make sure the strategy for the performance through giving its algorithm and investigating the source code. We also survey the algorithms of cylindrical algebraic decomposition to understand the NLSAT implementation.

Chapter 2

Polynomial

This chapter and the next chapter are mathematical preliminaries for Cylindrical Algebraic Decomposition.

In this chapter, we first see the algebraic structures such as Ring, Ideal, Field. Then, we see Primitive Element Theorem using the algebraic structures. The theorem is used in the Lift phase of Cylindrical Algebraic Decomposition.

The general reference here is, chapter 5 and Appendix.A in [9], chapter 2, 3, 4 in [10], appendix in [11], chapter II, IV, V in [12], chapter 9 in [13].

2.1 Ring

We first introduce Ring. Ring is a structure only allowed addition, subtraction and multiplication. We give here the definition.

Definition 2.1. A **ring** R is a set, together with two binary operations $\cdot$(multiplication) and $+$ (addition) on R satisfying the following conditions.

(i) $\forall a, b, c \in R \ ((a + b) + c = a + (b + c) \ \land \ (a \cdot b) \cdot c = a \ (b \cdot c))$ (associative).

(ii) $\forall a, b \in R \ (a + b = b + a)$ (commutative).

(iii) $\forall a, b, c \in R \ (a \cdot (b + c) = a \cdot b + a \cdot c)$ (distributive).

(iv) $\forall a \in R \ (\exists 0, 1 \in k \ (a + 0 = a \cdot 1 = a))$ (identities).

(v) $\forall a \in R \ (\exists b \in k \ (a + b = 0))$ (additive inverses).

Definition 2.2. A **commutative ring** R is a ring, satisfies multiplicative commutative condition,

(i) $\forall a, b \in R\ (a \cdot b = b \cdot a)$ (multiplicative commutative).

Definition 2.3. A **polynomial ring** $R[x]$ is set of mono variant polynomials whose coefficient is a ring.

2.2 Ideal

Ideal is a subset of a ring which is closed under addition and multiplication. We give the definition.

Definition 2.4. Given a ring R, a subset $I \subset R$ is an **ideal** if I satisfies the following conditions.

(i) $\forall a, b \in I\ (a + b \in I)$.

(ii) $\forall a \in I, b \in R\ (b \cdot a \in I)$.

Definition 2.5. Let $f \in R[x]$, we say the ideal $\{rf \mid r \in R[x]\}$ is a **principal ideal** generated by f, denotes $\langle f \rangle$.

Definition 2.6. Given a ring R, if all its ideals are principal ideal, we say it is a **principal ideal domain**, or the abbrev. **PID**.

Example 2.1. *The integer ring $\mathbb{Z}$ and a polynomial ring $k[x]$ in k, are both Principal Ideal Domain (PID).*

2.3 Polynomial

Polynomial is the main structure in this research report. At the same time, polynomial (polynomial ring) is an instance of Ring.

Definition 2.7. A **polynomial** f of variable x, written $f(x)$ is such that

$$f(x) = c_0 x^m + c_1 x^{m-1} + \cdots + c_{m-1} x_1 + a_m$$

where each $c_0, \ldots, c_m$ we say a **coefficient**. We say the each product pair of coefficient and variables $c_0 x^m, \ldots, c_1 x^{m-1}$ a **term**.

Example 2.2. $f(x) = 2x^2 + 3x + 4y$ *is a polynomial.* $2, 3, 4y$ *are coefficients of* $f(x)$. $2x^2, 3x, 4y$ *are terms of* $f(x)$.

Definition 2.8. A polynomial f is a **n-variant polynomial in integer** $\mathbb{Z}$, written $f \in \mathbb{Z}[x_1, \dots, x_n]$ such that

$$f(x_1, \dots, x_n) = c_{n_0} x_n^k + c_{n_1} x_n^{k-1} + \cdots + c_{n_i} x_n^{k-i} + \cdots + c_{n_{k-1}} x_n + c_{n_k} \quad (2.1)$$

where c_{n_i} are $n-1$-variant polynomials such that $c_{n_i}(x_1, \dots, x_{n-1}) = c_{n-1_0} x_{n-1}^l + c_{n-1_1} x_{n-1}^{l-1} + \dots c_{n-1_l}$. If $n = 1$, where c_{1_i} are mono variant polynomials with coefficients in integer such that $c_{1_i}(x_1) = c_{0_0} x_1^m + c_{0_1} x_1^{m-1} + \cdots + c_{0_m}$ where c_{0_i} are integers $\mathbb{Z}$.

Definition 2.9. Given a polynomial f of variable x,

$$f = c_0 x^m + c_1 x^{m-1} + \cdots + c_m,$$

We say $c_0 x^m$ that is the highest degree term of x in f, is the **leading term** of f, written $\mathtt{LT}_x(f) = c_0 x^m$.

Definition 2.10. Given a polynomial $f = c_0 x^m + c_1 x^{m-1} + \cdots + c_m$, we writes the coefficient of the leading term of f, $\mathtt{COEFF}_x(F)$.

Definition 2.11. Given n degrees polynomial f of variable x, we writes $\mathtt{deg}_x(f) = n$ is the degree of the polynomial. Specially, we define $\mathtt{deg}_x(0) = -\infty$. If the polynomial contains mono or no variable, we also writes $\mathtt{deg}(f) = n$ omitting the subscript variable x.

Example 2.3. *Let polynomials* f, g, p *are,*

$$\begin{aligned} f = x^3 + 2x^2 + x + 1, g &= 3y^2 + 3, \\ p = 0. \end{aligned}$$

Then, then the degree of f *is* 3*, the degree of* g *is* 2*, the degree of* p *is* $-\infty$*. We write* $\mathtt{deg}(f) = 3$*,* $\mathtt{deg}_y(g) = 2$*,* $\mathtt{deg}(p) = -\infty$*.*

2.4 Field

Field is a structure which allows division in each element, in addition to the operations of the ring. We give the definition.

Definition 2.12. A **field** k is a set, together with two binary operations $\cdot$(multiplication) and $+$(addition) on k satisfying the following conditions.

(i) $\forall a, b, c \in k \ ((a + b) + c = a + (b + c) \ \wedge \ (a \cdot b) \cdot c = a \ (b \cdot c))$ (associative).

(ii) $\forall a, b, c \in k\ (a + b = b + a\ \wedge\ a \cdot b = b \cdot a)$ (commutative).

(iii) $\forall a, b, c \in k\ (a \cdot (b + c) = a \cdot b + a \cdot c)$ (distributive).

(iv) $\forall a \in k\ (\exists 0, 1 \in k\ (a + 0 = a \cdot 1 = a))$ (identities).

(v) $\forall a \in k\ (\exists b \in k\ (a + b = 0))$ (additive inverses).

(vi) $\forall a \in k, a \neq 0\ (\exists c \in k\ (a \cdot c = 1))$ (multiplicative inverse).

2.5 Extension Field

If F is a subfield of E, we say E is an **extension field** of F. In this section, we introduce how to construct an extension field, through the definitions and the propositions.

Definition 2.13. Let $k[x]$ be a polynomial ring, $I \subset k[x]$ be an ideal. Given $f \in k[x]$, we say the **equivalence class of f with congruence modulo** I is,

$$[f] = \{g \in k[x] \mid f - g \in I\},$$

which is denoted $[f]$.

Definition 2.14. Let $I \subset k[x]$ be an ideal, the **quotient ring** of $k[x]$ modulo I, is the set of equivalence class for congruence modulo I, which is denoted $k[x]/I$:

$$k[x]/I = \{[f] \mid f \in k[x]\}.$$

Proposition 2.15. *Let $f \in k[x]$, set $\langle f \rangle$ be the principle ideal generated by f, if the f is irreducible, quotient ring $k[x]/\langle f \rangle$ forms a field.*

Proof. We say the quotient ring satisfies the field condition (2.12). For the detail, see proposition 4.4 in [10]. □

Proposition 2.16. *Let $f \in k[x]$ and f is irreducible, α is a root of $f(x)$, the quotient ring $k[x]/\langle f \rangle$ forms an **extension field** of k by α which denotes $f(\alpha)$,*

$$f(\alpha) = k[x]/\langle f \rangle.$$

Proof. See Lemma A.24 in [11], proposition 3.12 in [10]. □

2.6 Minimal Polynomial

Definition 2.17. α is an **algebraic number** over a field k if

$$\exists f(x) \in k[x] \ (f(\alpha) = 0).$$

Definition 2.18. Let α be a algebraic number over a field k. The **minimal polynomial** of α is the mono variant polynomial $f(x) \in k[x]$ of lowest degree such that $f(\alpha) = 0$.

2.7 Primitive Element Theorem

Finally, we see primitive element theorem which is used in the lift phase of Cylindrical Algebraic Decomposition.

Definition 2.19. Let α be a algebraic number, let $\mathbb{Q}$ be the ring of rational number. Let $Q(\alpha)$ be the minimal extension field of $\mathbb{Q}$ including α. We say $\mathbb{Q}(\alpha)$ is the simple extension on $\mathbb{Q}$, and the α is the **primitive element** of $\mathbb{Q}(\alpha)$.

Theorem 2.20. *(Primitive Element Theorem) Let α, β be algebraic numbers on $\mathbb{Q}$. Then,*

$$\exists \gamma \ (\mathbb{Q}(\gamma) = \mathbb{Q}(\alpha, \beta)),$$

the γ is the primitive element of $\mathbb{Q}(\alpha, \beta)$.

Proof. See THEOREM 26,27 in [14], Appendix.1 in [15]. □

Algorithm of Primitive Elements and its minimal polynomial

We give an algorithm to get minimal polynomial for extension field. At the same time, this algorithm is used in Lift phase of Cylindrical Algebraic Decomposition. The reference of this algorithm is [16–18].

In the algorithm, we need square free polynomial.

Definition 2.21. Let a polynomial $P(x) = (x - \alpha_1)^{e_1}(x - \alpha_2)^{e_2} \dots (x - \alpha_n)^{e_n}$ where α_n are the roots of $f(x)$, e_n are the multiplicity of the roots. Then, the square free $f(x)$ of $P(x)$ is

$$f(x) = (x - \alpha_1)(x - \alpha_2) \dots (x - \alpha_n).$$

Furthermore, it is defined by $f(x)$ such that $\mathrm{GCD}(f, f') = 1$, where f' is the derivative of f.

In other words, the square free is the polynomial that multiple roots are removed from the factorization.

To calculate the square free we give SQFree algorithm.

Algorithm 1 SQFree(P)

INPUT: $P = (x-\alpha_1)^{e_1}(x-\alpha_2)^{e_2}\dots(x-\alpha_n)^{e_n}$
OUTPUT: $f = (x-\alpha_1)(x-\alpha_2)\dots(x-\alpha_n)$

$f := P$
while $h = \mathrm{GCD}(f, f') \neq 1$ **do**
  $f := h$
end while
return f

Proof. We use the property that $\mathrm{GCD}(P, P')$ reduce the number of the multiplicity of the roots. For instance, let $P(x) = (x+\alpha_1)^2(x+\alpha_2)^3$. From the product rule, the derivative of $P(x)$ be,

$$\begin{aligned} P' &= 2(x+\alpha_1)(x+\alpha_2)^3 + (x+\alpha_2)^2 3(x+\alpha_2)^2 \\ &= (x+\alpha_1)(x+\alpha_2)^2(2(x+\alpha_2)^2 + 3(x+\alpha_1)(x+\alpha_2)). \end{aligned}$$

This means $\mathrm{GCD}(P, P') = (x+\alpha_1)(x+\alpha_2)^2$ whose multiplicity of roots e_i are $e_i = e_i - 1$. We continue the calculation in the while loop without loss of generality, finally we get the square free where all $e_i = 1$ when $\mathrm{GCD}(f, f') = 1$. □

Theorem 2.22. *Let $g(x,\alpha)$ be the minimal polynomial for an algebraic number β over $\mathbb{Q}(\alpha)$, and $f(y)$ be the minimal polynomial for an algebraic number α over $\mathbb{Q}$. If h that is the power of $g(x,\alpha)$ is square free, then $\mathbb{Q}(\alpha,\beta) = \mathbb{Q}(\beta)$, and $h(x)$ in $\mathbb{Q}$ as the minimal polynomial of $g(x,\alpha)$.*

Proof. See Theorem 3.2 in [17], Lemma 3.1 in [18], Lemma 3.2 in [16] □

As a constructive proof of the theorem (2.22), we give the following algorithm. Here, we use the $\mathtt{Resultant}(f, g)$ in (3.4), to calculate the power of $g(x,\alpha)$.

Algorithm 2 MinPolByPrimElem(α, β)

INPUT: $\alpha = f(y)$ over $\mathbb{Q}$, $\beta = g(x, \alpha)$ over $\mathbb{Q}(\alpha)$
OUTPUT: the minimal polynomials for γ where $\mathbb{Q}(\alpha, \beta) = \mathbb{Q}(\gamma)$, and for β $h(x)$ over $\mathbb{Q}$

$s := 0$
$h(x) = Resultant_y(f(y), g(x, y))$
while $\mathsf{GCD}(h(x), h(x)') \neq 1$ **do**
  $s := s + 1$
  $h(x) = Resultant_y(f(y), g(x - sy, y))$
end while
$\gamma = s\alpha + \beta$
return γ, $h(x)$

In the while loop, we calculate $g(x - sy, y)$, if $\alpha = y^2 - 2$, $\beta = x^2 + y^2 + 4$, and $s = 1$, then from $g(x - sy, y)$, we get $(x - y)^2 + y^2 + 4$.

Chapter 3

Greatest Common Divisor

In this chapter, we introduce the theorems and algorithms related to Greatest Common Divisor for Cylindrical Algebraic Decomposition.

The reference of the definitions, theorems and the proofs in this chapter are Chapter 1 in [9], Chapter 4, 5 in [19], Chapter 2 in [20], Chapter 9 in [13], Chapter 3 in [21].

3.1 Greatest Common Divisor

We first define greatest common divisor in this section.

Definition 3.1. The polynomial q is a **divisor** of a polynomial f if $f = aq$ for some polynomial a.

Definition 3.2. A polynomial h is the **greatest common divisor** of two polynomials f and g which is denoted $\mathtt{GCD}(f, g)$, if h is a divisor of both f and g, and all other divisors of both f and g are divisor of h.

Example 3.1. *Let* f, g, $f(x) = x^3 - 6x^2 + 11x - 6$, $g(x) = x^2 - 2x + 1$. *Then* $h = \mathtt{GCD}(f, g) = x - 1$ *is the greatest common divisor of* g *and* f. *Since the factorization of* f, g *is* $f(x) = (x-1)(x-2)(x-3)$, $g(x) = (x-1)^2$, *thus* $h = x-1$ *is the greatest common divisor.*

3.2 Euclidean Algorithm

Given polynomials $f(x)$ and $f_1(x)$, where $\mathtt{deg}(f) \geq \mathtt{deg}(f_1)$, then divide $f(x)$ with $f_1(x)$. We write $f(x)$ by the following equation,

$$f(x) = q(x)f_1(x) + f_2(x), \tag{3.1}$$

where $q(x)$ is the quotient, $f_1(x)$ is the divisor, $f_2(x)$ is the reminder. In polynomial division, the reminder has always lower degree than the divisor. We say $\mathtt{deg}(f_i) > \mathtt{deg}(f_{i+1})$. So here it is $\mathtt{deg}(f_1) > \mathtt{deg}(f_2)$.

Continuously, we write,

$$f_1(x) = q_1(x)f_2(x) + f_3(x) \tag{3.2}$$
$$f_2(x) = q_2(x)f_3(x) + f_4(x) \tag{3.3}$$
$$\dots \tag{3.4}$$
$$f_{k-1}(x) = q_k(x)f_k(x) + 0 \tag{3.5}$$

Since the degree of reminder is always lower than the divisor, so that the procedure is stopped when the reminder is 0 at (3.5).

Now, we prove the following proposition.

Proposition 3.3. *Given the above polynomials $f(x)$, $f_1(x)$, and the reminders $f_2, \dots, f_{k-1}$, and the last divisor f_k, then we say*

$$\mathtt{GCD}(f, f_1) = \mathtt{GCD}(f_1, f_2) = \dots = \mathtt{GCD}(f_k, 0) = f_k.$$

Proof. The crucial equation in the above proposition is,

$$\mathtt{GCD}(f, f_1) = \mathtt{GCD}(f_1, f_2). \tag{3.6}$$

First, we say $\mathtt{GCD}(f, f_1) \implies \mathtt{GCD}(f_1, f_2)$.

Let $h = \mathtt{GCD}(f, f_1)$, then h is a common divisor of f and f_1 by the definition of greatest common divisor (3.2). From the equation (3.1), we write

$$\begin{aligned} f_2(x) &= f(x) - q(x)f_1(x) \\ &= (Ah) - q(x)(Bh) \\ &= h(A - q(x)B). \end{aligned}$$

Thus h is also a divisor of f_2. So h is a common divisor of f_1 and f_2. Furthermore, there is no other common divisor of f_1 and f_2 greater than h, because if such divisor $h' = \mathtt{GCD}(f_1, f_2)$ exists, the h' is also a divisor of f, then $h' = \mathtt{GCD}(f, f_1)$. However it contradicts how we took $h = \mathtt{GCD}(f, f_1)$. Thus h is the greatest, and $h = \mathtt{GCD}(f_1, f_2)$. So $\mathtt{GCD}(f, f_1) \implies \mathtt{GCD}(f_1, f_2)$.

Next, we say the opposite direction $\mathtt{GCD}(f_1, f_2) \implies \mathtt{GCD}(f, f_1)$.

If $h_2 = \mathtt{GCD}(f_1, f_2)$, the h_2 is a divisor of f, because

$$\begin{aligned} f(x) &= q(x)f_1(x) + f_2(x) \\ &= q(x)Ah_2 + Bh_2 \\ &= h_2(q(x)A + B). \end{aligned}$$

So $\mathtt{GCD}(f_1, f_2) \implies \mathtt{GCD}(f, f_1)$. Thus, it is proved that $\mathtt{GCD}(f, f_1) = \mathtt{GCD}(f_1, f_2)$.

We apply the same procedure on $\mathtt{GCD}(f_1, f_2), \dots, \mathtt{GCD}(f_k, 0)$. So the remaining to show is $\mathtt{GCD}(f_k, 0) = f_k$. This is obvious from the definition of greatest common divisor. Thus,

$$\mathtt{GCD}(f, f_1) = \mathtt{GCD}(f_1, f_2) = \cdots = \mathtt{GCD}(f_k, 0) = f_k.$$

□

From the proposition, we give an algorithm to calculate the greatest common divisor.

Algorithm 3 Euclidean(f, f_1)

```
INPUT: f, f1 are polynomial
OUTPUT: h is the greatest common divisor of the given polynomials
h := f
d := f1
while d ≠ 0 do
    r := h − Qd                                  ▷ Find Q and calculate r
    h := d
    d := r
    return h
end while
```

Example 3.2. *Given* $f = x^3-6x^2+11x-6$, $f_1 = x^2-2x+1$, Euclidean(f, f_1) *calculates the following. In the first loop:*

$$\begin{aligned}
h &= x^3 - 6x^2 + 11x - 6\\
d &= x^2 - 2x + 1\\
r &= (x^3 - 6x^2 + 11x - 6) - (x-4)(x^2 - 2x + 1) = 2(x-1)\\
h &= x^2 - 2x + 1\\
d &= 2(x-1).
\end{aligned}$$

In the second loop,

$$\begin{aligned}
h &= x^2 - 2x + 1\\
d &= 2(x-1)\\
r &= x^2 - 2x + 1 - \frac{1}{2}(x-1)2(x-1) = x^2 - 2x + 1 - (x-1)(x-1) = 0\\
h &= x - 1\\
d &= 0.
\end{aligned}$$

Then it stops since $d = 0$, *and returns* Euclidean$(f, f_1) = h = x - 1$.

3.3 Resultant

We introduce resultant as a condition of a common root.

Definition 3.4. The **resultant** denoted $\mathsf{Resultant}_x(f, g)$ of the two polynomials

$$\begin{aligned}
f(x) =& a_0x^m + a_1x^{m-1} + \cdots + a_m\\
g(x) =& b_0x^n + b_1x^{n-1} + \cdots + b_n
\end{aligned}$$

where a_0 and b_0 are not 0, is a polynomial in the a_i and b_i, such that $\mathsf{Resultant}_x(f, g) = 0$ if and only if f and g have a common root.

Remark 3.5. We also denote $\mathsf{Resultant}(f, g)$ omitting the subscript x, if the variable is obvious.

We give here how to construct $\mathsf{Resultant}(f, g)$.

Let the root of f be $\alpha_1 \ldots \alpha_m$, and the root of g be $\beta_1 \ldots \beta_m$. Then write f and g :

$$f(x) = a_0(x-\alpha_1)(x-\alpha_2)\ldots(x-\alpha_m), \tag{3.7}$$

$$g(x) = b_0(x-\beta_1)(x-\beta_2)\ldots(x-\beta_n). \tag{3.8}$$

Based on this, we define $\mathsf{Resultant}(f, g)$ as the following three equivalent forms,

$$\mathsf{Rresultant}(f, g) = a_0^n b_0^m (\alpha_1 - \beta_1) \dots (\alpha_m - \beta_n) \tag{3.9}$$
$$= b_0^m g(\alpha_1) \dots g(\alpha_m) \tag{3.10}$$
$$= (-1)^{mn} f(\beta_1) \dots f(\beta_n). \tag{3.11}$$

From (3.9), we say $\mathsf{Resulant}(f, g) = 0$ if and only if f and g have a common root. Because if any α_i and β_j are the same, the right hand side must be 0, else it is never become 0. So $\mathsf{Resultant}(f, g) = 0$.

Then, we prove that the above $\mathsf{Resultant}(f, g)$ is written in the coefficients a_i and b_i.

Proposition 3.6. $\mathsf{Resultant}(f, g)$ *is written in the coefficients of the given two polynomials.*

Proof. From (3.10) or (3.11), $\mathsf{Resultant}(f, g)$ be a symmetric function of α_i or β_i, and symmetric function is represented by the elementary symmetric polynomials. Moreover, elementary symmetric polynomials is represented by the coefficients of the given polynomial:

$$\alpha_1 + \alpha_2 + \dots \alpha_m = -a_1,$$
$$\alpha_1\alpha_2 + \dots \alpha_{m-1}\alpha_m = a_2,$$
$$\vdots$$
$$\alpha_1\alpha_2 \dots \alpha_m = (-1)^m a_m.$$

Thus $\mathsf{Resultant}(f, g)$ is written in the coefficients of the given two polynomials. □

From the discussion on (3.9) and the proposition (3.6), our construction of $\mathsf{Resultant}(f, g)$ satisfies the resultant definition.

Sylvester Matrix

We introduce the definition of sylvester matrix as a matrix representation of resultant. Thanks to the matrix representation, we can calculate the resultant efficiently.

Definition 3.7. Let polynomials f, g be,

$$f = a_m x^m + a_{m-1} x^{m-1} + \dots a_0, \tag{3.12}$$
$$g = b_n x^n + b_{n-1} x^{n-1} + \dots b_0. \tag{3.13}$$

Then the **sylvester matrix** is a $m+n$ matrix which denoted Sylvester(f, g) such that

$$\text{Sylvester}(f,g) = \begin{pmatrix} a_m & a_{m-1} & \dots & a_1 & a_0 & 0 & 0 & 0 \\ 0 & a_m & a_{m-1} & \dots & a_1 & a_0 & 0 & 0 \\ \vdots & & \ddots & & & \ddots & \ddots & 0 \\ 0 & 0 & 0 & a_m & a_{m-1} & \dots & a_1 & a_0 \\ b_n & b_{n-1} & \dots & b_1 & b_0 & 0 & 0 & 0 \\ 0 & b_n & b_{n-1} & \dots & b_1 & b_0 & 0 & 0 \\ \vdots & & \ddots & & & \ddots & \ddots & \\ 0 & 0 & 0 & b_n & b_{n-1} & \dots & b_1 & b_0 \end{pmatrix}.$$

We say the next theorem with sylvester matrix.

Theorem 3.8. *The determinant of Sylvester*(f, g) *is equal to* $\mathtt{Resultant}(f, g)$.

To prove the theorem, we first prove the following lemma.

Lemma 3.9. *Given two polynomial* f, g *where* $\deg(f) \geq \deg(g)$*, then there is a common factor if and only if* $Af + Bg = 0$ *where* A *and* B *are nonzero polynomials, and those degrees be* $\deg(A) \leq \deg(g) - 1$*,* $\deg(B) \leq \deg(f) - 1$*.*

Proof. Assume h is a common factor of f and g, we write,

$$\begin{aligned} f &= hf', \\ g &= hg'. \end{aligned}$$

Now let $A = g'$, $B = -f'$, then,

$$\begin{aligned} Af + Bg &= g'hf' + (-f')g \\ &= gf' - gf' \\ &= 0. \end{aligned}$$

Thus we say if there is a common factor, $Af + Bg = 0$.

Conversely, if $Af + Bg = 0$, we write $Bg = -Af$. So g is a divisor of the left hand side, also the right hand side must be divided by g in the equation. At the same time $\deg(A) \leq \deg(g) - 1$ from the condition, meaning that A must not have all the factor of g. So f must contains a factor of g to hold the equation. So if $Af + Bg = 0$, there is a common factor of f and g. □

Using this lemma, we prove the theorem (3.8).

Proof. Let f, g be (3.12), (3.13) respectively, and A, B be,

$$A(x) = s_{n-1}x^{n-1} + s_{n-2}x^{n-1} + \cdots + s_0, \tag{3.14}$$
$$B(x) = t_{m-1}x^{m-1} + t_{m-2}x^{n-1} + \cdots + t_0. \tag{3.15}$$

Then, we calculate $Af + Bg = 0$, and we compare the coefficients of power of x, we get the following $m + n$ unknowns, $m + n$ system of linear equations.

$$\begin{cases} a_m s_{n-1} + b_n t_{m-1} & = 0, \\ a_{m-1}s_{n-1} + a_m s_{n-1} + b_{n-1}t_{m-1} + b_n t_{m-1} & = 0, \\ & \vdots \\ a_0 s_0 + b_0 t_0 & = 0. \end{cases} \tag{3.16}$$

Thus it is expressed as the following matrix representation,

$$(s_{n-1}, \ldots, s_0, t_{m-1}, \ldots, t_0)\text{Sylvester}(f, g) = 0 \tag{3.17}$$

At the same time, having a nonzero solution of a linear equation is equal to the coefficient matrix determinant is 0. The determinant is exactly the determinant of Sylvester(f, g). Also the system of liner equation (3.17) represents $Af + Bg = 0$. Consequently, using the lemma (3.9), the determinant of Sylvester matrix satisfy the resultant definition (3.4). □

3.4 Sub-resultant Coefficient

In this section, we introduce sub-resultant coefficient as a representation of extended euclidean algorithm.

3.4.1 Extended Euclidean Algorithm

When calculating Greatest Common Divisor we say the following identity.

Theorem 3.10. *Given polynomials* $f(x), g(x)$. *Let* $h = \mathtt{GCD}(f, g)$. *Then there exists some polynomials* A, B *such that*

$$h = Af + Bg \tag{3.18}$$

We introduce **Extended Euclidean Algorithm** as a constructive proof of this theorem.

Extend Euclidean Algorithm calculates the following a_i, b_i besides euclidean algorithm calculates the greatest common divisor. The algorithm starts $f_0 = f, f_1 = g$, $a_0 = 1$, $a_1 = 0$, $b_0 = 0$, $b_1 = 1$.

$$\begin{cases} f_{i-1}(x) &= q_i(x)f_i(x) + f_{i+1}(x) \\ a_{i+1}(x) &= a_{i-1}(x) - q_i(x)a_i(x) \\ b_{i+1}(x) &= b_{i-1}(x) - q_i(x)b_i(x) \end{cases} \tag{3.19}$$

As we see in (3.5), this calculation stops when the reminder is zero,

$$f_{k-1}(x) = q_k(x)f_k(x) + 0.$$

As the result, $f_k = \mathtt{GCD}(f, g)$. Furthermore, let c be the principal coefficient of f_k, **Extended Euclidean Algorithm** calculates the A and B as $A = a_k$, $B = b_k$.

We prove the theorem (3.10) with this algorithm.

Proof. We prove the following identity by mathematical induction on i. In each step of Extended Euclidean Algorithm, the equation

$$f_i(x) = a_i(x)f(x) + b_i(x)g(x) \tag{3.20}$$

holds.

When $i = 1$, the equation is,

$$f_1(x) = a_1(x)f(x) + b_1(x)g(x).$$

From $f_1 = g, a_1 = 0, b_1 = 1$, we rewrite the equation,

$$\begin{aligned} g(x) &= 0f(x) + 1g(x) \\ g(x) &= g(x). \end{aligned}$$

So the claim hold on $i = 1$.

When $i = 2$, we say a_2 and b_2 be,

$$\begin{aligned} a_2(x) &= a_0(x) - q_1(x)a_1(x) \\ &= 1 - q_1(x)0 \\ &= 1 \end{aligned} \tag{3.21}$$

$$
\begin{aligned}
b_2(x) &= b_0(x) - q_1(x)b_1(x) \\
&= 0 - q_1(x)1(x) \\
&= q_1(x).
\end{aligned} \tag{3.22}
$$

From $f_0 = f, f_1 = g$, and (3.19), (3.21), (3.22), we rewrite f_2 as,

$$
\begin{aligned}
f_0(x) &= q_1(x)f_1(x) + f_2(x) \\
f_2(x) &= f_0(x) - q_1(x)f_1(x) \\
&= f(x) - q_1(x)g(x) \\
&= 1f(x) - q_1(x)g(x) \\
&= a_2(x)f(x) - b_2(x)g(x).
\end{aligned}
$$

So the equation holds when $i = 2$.

Let us assume the claim holds until $i > 2$. Then $i + 1$ be,

$$
\begin{aligned}
f_{i+1} &= f_{i-1}(x) - q_i(x)f_i(x) \\
&= (f(x)a_{i-1} + g(x)b_{i-1}(x)) - q_i(x)(f(x)a_i(x) + g(x)b_i(x)) \\
&= f(x)(a_{i-1}(x) - q_i(x)a_i(x)) + g(x)(b_{i-1}(x) - q_i(x)b_i(x)) \\
&= f(x)a_{i+1} + g(x)b_{i+1}.
\end{aligned}
$$

By mathematical induction on i, the identity holds for all i. So the equation (3.20) holds on $f_k = GCD(f, g) = h$, thus there exits polynomial A, B such that $h = Af + Bg$. □

3.4.2 Sub-resultant coefficient

In the previous section, we see extended euclidean algorithm which calculates $GCD(f, g) = Af+Bg$. We defines j-th sub-resultant coefficient as a representation of the greatest common divisor calculation.

Definition 3.11. We define **j-th sub-resultant coefficient** from the equation of extended euclidean algorithm (3.19). We write f, g, h, A, B be,

$$
\begin{aligned}
f &= a_m x^m + \cdots + a_1 x + a_0, \\
g &= b_n x^n + \cdots + b_1 x + b_0, \\
h &= c_j x^j + \cdots + c_1 x + c_0, \\
A &= s_{n-j-1} x^{n-j-1} + \cdots + s_1 x + s_0, \\
B &= t_{m-j-1} x^{m-j-1} + \cdots + t_1 x + t0
\end{aligned}
$$

Then from the equation $h = Af + Bg$ (3.18), and comparing the coefficients of power of x, we get the following $m + n - 2j$ unknowns, $m + n - 2j$ system of linear equations.

$$\begin{cases} s_{n-j-1}a_m + t_{m-j-1}b_n & = 0, \\ s_{n-j-1}a_{m-1} + s_{n-j-2}a_m + t_{m-j-1}b_{n-1} + t_{m-j-2}b_n & = 0, \\ \quad\vdots & \\ s_j a_0 + s_{j-1}a_1 + \cdots + s_0 a_j + t_j b_0 + t_{j-1}b_1 + \ldots t_0 b_j & = c_j \end{cases} \tag{3.23}$$

Remark 3.12. Why think $m + n - 2j$ unknown is because the j-th degree is essential to think $\mathtt{GCD}(f, g)$, the lower $j - 1$-th to 0-th degrees coefficients are removed from the systems of linear equations.

Thinking in matrix representation we get the following W and R_j.

$$W = (s_{n-j-1}, \ldots, s_0, t_{m-j-1}, \ldots, t_0)$$

$$R_j(f,g) = \left(\begin{array}{ccccccccc} a_m & a_{m-1} & \ldots & a_j & a_{j-1} & \ldots & a_1 & a_0 & 0 \\ 0 & a_m & a_{m-1} & \ldots & a_j & a_{j-1} & \ldots & a_1 & a_0 \\ \vdots & & \ddots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & a_m & \ldots & \ldots & a_j & a_{j-1} & \vdots \\ 0 & 0 & 0 & 0 & a_m & \ldots & & a_j & a_{j-1} \\ 0 & 0 & 0 & 0 & 0 & a_m & \ldots & \ldots & a_j \\ b_n & b_{n-1} & \ldots & b_j & b_{j-1} & \ldots & b_1 & b_0 & 0 \\ 0 & b_n & b_{n-1} & \ldots & b_j & b_{j-1} & \ldots & b_1 & b_0 \\ \vdots & & \ddots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & b_n & b_{n-1} & \ldots & b_j & b_{j-1} & \ldots \\ 0 & 0 & 0 & 0 & b_n & b_{n-1} & \ldots & b_j & b_{j-1} \\ 0 & 0 & 0 & 0 & 0 & b_n & b_{n-1} & \ldots & b_j \end{array}\right) \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array}\right\} n-j \\ \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array}\right\} m-j \end{array}$$

Then we write $Af + Bg = h$ (3.18) as,

$$WR_j(f, g) = (0, \ldots, 0, c_j) \tag{3.24}$$

We say the determinant of $R_j(f, g)$ as **j-th sub-resultant coefficient** denotes $\mathrm{PCS}_j(f, g)$.

3.4.3 The degree of Greatest Common Divisor

Using the j-th sub-resultant coefficient, we get the degree of greatest common divisors without calculating the $\mathtt{GCD}$ itself.

Theorem 3.13. *Given polynomial* f, g *where* $\deg(f) = m$, $\deg(g) = n$.

For all $0 \leq l < j$,

$$\deg(\mathtt{GCD}(f, g)) = l \Leftrightarrow \mathsf{PSC}_j(f, g) = 0 \wedge \mathsf{PSC}_l(f, g) \neq 0. \tag{3.25}$$

Proof. Here we sketch the proof.

First we say $\deg(\mathtt{GCD}(f, g)) = l \Rightarrow \mathsf{PSC}_j(f, g) = 0 \wedge \mathsf{PSC}_l(f, g) \neq 0$.

If there is a solution on W of 3.24, the $\deg(Af + Bg) = j$, and the $\mathtt{GCD}(f, g)$ divides $Af + Bg$. So $l = \deg(h) \leq (\deg(Af + Bg)) = j$.

This means $\mathsf{PSC}_j(f, g) \neq 0 \Rightarrow l \leq j$. The contraposition is for all j where $0 \leq j < l \Rightarrow \mathsf{PSC}_j(f, g) = 0$.

Next we say $\mathsf{PSC}_l(f, g) \neq 0$. If $\mathsf{PCS}_l(f, g) = 0$, it contradict how we construct $WR_l(f, g) = (0, 0, \ldots, c_l)$ in 3.24 so $\mathsf{PCS}_l(f, g) \neq 0$.

Last we say opposite direction, For all l, $0 \leq l < j$, $\mathsf{PSC}_j(f, g) = 0 \wedge \mathsf{PSC}_l(f, g) \neq 0 \Rightarrow \deg(\mathtt{GCD}(f, g)) = l$.

From how we construct $WR_j(f, g)$, $\mathsf{PSC}_{\deg(\mathtt{GCD}(f,g))}(f, g) \neq 0$, so $l \leq \deg(\mathtt{GCD}(f, g))$. Since $\mathtt{GCD}(f, g)$ divides $Af + Bg$, we say $\deg(\mathtt{GCD}(f, g)) \leq \deg(Af + Bg) = l$. Thus we say $\deg(\mathtt{GCD}(f, g)) = l$. □

The detailed proofs can be found in Theorem 3.1 in [21], and another way of proof using the Least Common Multiple of f and g is found in proposition 4.25 in [22].

Chapter 4

SAT/SMT solver

This chapter is a preliminary of NLSAT that is a SMT solver specialized to non linear real arithmetic.

Here we introduce the ideas of SAT, SMT solvers and its algorithms. We assume basic knowledge of propositional logic and predicate logic in this chapter.

4.1 SAT Solver

In this section, we first give the definitions related to SAT solver. Then, introduce an algorithm of SAT solvers.

Definition 4.1. We say a **Satisfiability Problem** is to decide if there is an assignment to make the given CNF formula satisfiable (SAT), or not (UNSAT)

Definition 4.2. We say a set is **CNF** if it consists of conjunctive clauses whose each clauses consists of a disjunction of literals.

Example 4.1. $\varphi = (A \vee B \vee C) \wedge (\neg A \vee C) \wedge (\neg B \vee C \vee D)$
where φ *is a CNF.*

The above example is SAT. $A \wedge \neg B \wedge C$ is an satisfiable assignment.

From now on, we writes CNF by set for convenience. We writes a set of literals $\{l_1, l_2, l_3, l_4\}$ as the clause $(l_1 \vee l_2 \vee l_3 \vee l_4)$. Moreover, we writes a set of clauses $\{C_1, C_2, C_3\}$ as a conjunctive normal form $C_1 \wedge C_2 \wedge C_3$. So we can write the above CNF by the set $\{\{A, B, C\}, \{\neg A, C\}, \{\neg B, C, D\}\}$

Definition 4.3. SAT solver is a program to solve Satisfiability Problem.

DPLL Algorithm

In this section, we introduce an algorithm named DPLL which solves Satisfiability Problem with the following techniques.

Depth First Search in Binary Decision Tree: DPLL algorithms searches the satisfiable assignments in the binary decision tree of all the literals. The tree height is at most n where n is the number of literals. It recursively decides an assignment of literal which is either True or False. Each time it decides an assignment, it tests whether the current assignments satisfy the given CNF. If un-satisified, it backtracks the tree.

Unit-Resolution: Unit-Resolution (or Unit Propagation) is a way to apply an inference rule. It picks the literal from the unit clauses as the decided literal.

Conflict-Driven Clause Learning (CDCL): CDCL adds a new clause into the original CNF when DPLL detects a conflict so that it will realize early the decision is not satisfiable.

Non-Chronological Backtracking: Non-Chronogical Backtracking is a backtracking which backtracks previous wrong decision. As the result it backtracks more than one step in the tree using the information of the conflicts called **Implication Graph**.

Definition 4.4. If a clause contains only one literal, it is called a **unit clause**, also we denote by literal(unit clause) as the literal.

Example 4.2. $\{C\}, \{A\}, \{\neg B\}$ *are unit clauses.* $\{C, \neg A\}, \{B, A, C\}, \{\}$ *are not unit clauses.* $\mathit{literal}(\{C\}) = C$.

First, we give the algorithm of Unit Resolution to explain the pure functionality.

Algorithm 4 UnitResolution(a CNF φ)

INPUT: A CNF φ
OUTPUT: A set of decided literals D picked from unit clauses or derived from φ by resolution.

```
F := φ
D := ∅
for all C ∈ F ∧ C is a unit clause do
    l := literal(C)
    D := D ∪ {l}
    for all ¬l ∈ C ∈ F do
        C := C \ {¬l}                 ▷ Remove the literal because l ∨ ¬l
        if C is a unit clause then
            D := D ∪ {literal(C)}     ▷ Add the decision as the result of resolution
        end if
    end for
end for
return D
```

Example 4.3. *Let* $\varphi = \{\{A, \neg B\}, \{B\}, \{B, C\}, \{E, F\}\}$*, the result of UnitResolution(*φ*) is* $D = \{A, B\}$*. Let* $\varphi = \{\{\neg B, C\}, \{B\}$*, the result is* $D = \{B, C\}$

In DPLL algorithm, the conflicts are occurred inside UnitResolution, thus we extend the algorithm to do Conflict Driven Clause learning so that we can use it in DPLL algorithm.

Algorithm 5 UnitResolution+(a CNF φ, decision set D)

INPUT: A CNF φ, decision set D
OUTPUT: The decided literals D, the conflict reason clauses (assertion clauses) A.

```
F := φ
D := D ∪ { a set of literal of unit clauses in F}
A := ∅
i := 0                                   ▷ The clause index of given CNF
G := a dictionary                        ▷ We assume we have dictionary data structure

while D ≠ ∅ do
   l := D[i]
   for all ¬l ∈ C ∈ F do
      C := C \ {¬l}
      j := the index of C
      G[j] := G[j] ∪ {l}       ▷ We make a graph to know where come from the conflict
   end for
   if i is the last index of D then
      for all C ∈ F ∧ C is a unit clause do
         D := D ∪ {literal(C)}
      end for
   end if
   if i is the last index of D then
      return D, A
   end if
   i := i + 1
end while

if C ∈ F ∧ C = ∅ then
   j := the index of C
   A := G[j]
   return D, A                           ▷ We found a contradiction
end if
return D, A                              ▷ We come here when D = ∅
```

Finally, we give the DPLL algorithm using UnitResolution+ algorithm.

Algorithm 6 DPLL(a CNF φ)

INPUT: A CNF φ
OUTPUT: SAT or UNSAT, and if SAT, returns satisfiable assignments D, else returns $D := \emptyset$

$F = \varphi$
$D = \emptyset$ ▷ decided literals
$L = \emptyset$ ▷ L is learnt clauses from conflict

while $true$ **do**
  $D', A :=$ UnitResolution+$(F \cup L, D)$
  if $D' = D \wedge A \neq \emptyset$ **then**
    return UNSAT, $D := \emptyset$ ▷ It conflicts without any new decisions, meaning UNSAT
  end if
  D := D'
  if $A \neq \emptyset$ **then**
    $S :=$ the second clause from the last in A
    $s :=$ the index of S in D
    $D :=$ first s items in D ▷ Non-chronological Backtracking
    $L := L \cup \neg A$ ▷ Conflict-Driven Clause Learning
  else
    if $l \in C \in F \wedge \{l, \neg l\} \notin D$ **then**
      $D := D \cup \{l$ or $\neg l$ not in $D\}$ ▷ select l or $\neg l$ as the decision
    else
      return SAT, D
    end if
  end if
end while

4.2 SMT solver

In this section, we first give the definitions related to SMT solver. Then, we overview the SMT solvers approachs. Finally, give an algorithm for SMT solver.

Definition 4.5. We say a **Satisfiability Modulo Theories (SMT) Problem** is to decide if there is an assignment to make the given formulas with respect to some background theories expressed in first-order logic satisfiable (SAT), or not (UNSAT). Furthermore, we say **$\mathcal{T}$-formula** as the formulas.

Example 4.4. $x > y \wedge \neg(x < y - 2)$ *is a $\mathcal{T}$-formula. To decide the satisfiability is a SMT problem.*

Definition 4.6. SMT solver is a program to solve the SMT problem.

Next, we define **theory solver** for SMT solvers. SMT sovlers solve SMT problems with theory solvers.

Definition 4.7. A **theory solver** is a decision procedure which determines the given set of conjunction of formula conflicts or not in the theory $\mathcal{T}$.

4.2.1 Approach of SMT solvers

Here we overview the approaches of SMT solvers with the following three definitions.

Definition 4.8. Eager approach (or **Eager SMT Techniques** [23]) to SMT is translating $\mathcal{T}$-formula into a satisfiability preserved boolean CNF. Then check the boolean satisfiability with SAT solver.

Remark 4.9. The translation is different than boolean abstraction T2B (we see in next section). The Eager approach translations are for example, **per-constraint encoding** in [24], **small domain encoding** in [25] for Logic of Equality with Uninterpreted Functions (EUF).

Definition 4.10. Lazy approach (or **Lazy SMT Techniques** [23]) to SMT solver is using theory solver for conjunction of theory literals in a SMT solver.

Remark 4.11. Nieuwenhuis, Olivers, and Tinelli use the word **eager** and **lazy** to explain their DPLL(T) framework in [23]. In the paper, eager means using theory solvers early, lazy means using theory solvers late.

Definition 4.12. Splitting on demand [26] is a sub approach of Lazy approach. Splitting on demand is doing case splitting inside theory solvers internally. It decides both boolean literals and also its variables inside the theory literals. Both boolean literals and the variable inside theory literals are handled in the same engine.

4.2.2 A DPLL(T) algorithm

DPLL(T) is an extension of DPLL algorithm which employs theory solvers for SMT problems. Since DPLL(T) employs theory solver, it is a lazy approach SMT solver.

We give here an algorithm $\mathcal{T}$-DPLL it is a very lazy variation of the DPLL(T).

First, we defines the sub algorithms of $\mathcal{T}$-DPLL: T2B, B2T, and $\mathcal{T}$-Solver.

T2B is a algorithm to do boolean abstraction of $\mathcal{T}$-formula.

Algorithm 7 T2B(a $\mathcal{T}$-formula φ)

INPUT: A $\mathcal{T}$-formula φ
OUTPUT: A CNF of φ

$\mathsf{F} := \emptyset$
for all $\mathsf{C} \in \varphi$ **do**
  $\mathsf{C}' := \emptyset$
  for all $\mathsf{l} \in \mathsf{C}$ **do**
    if l is a formula **then**
      B := add a boolean variable expressed the formula l
      $\mathsf{C}' := \mathsf{C} \cup \mathsf{B}$
    else
      $\mathsf{C}' := \mathsf{C} \cup \mathsf{l}$
    end if
  end for
  $\mathsf{F} := \mathsf{F} \cup \mathsf{C}'$
end for

B2T is an opposite function of T2B.

Algorithm 8 B2T(a CNF φ)

INPUT: A CNF φ
OUTPUT: A $\mathcal{T}$-formula of φ

$F := \emptyset$
for all $C \in \varphi$ **do**
  $C' := \emptyset$
  for all $B \in C$ **do**
    if B is a boolean value expressed a formula f **then**
      $C' := C \cup f$
    else
      $C' := C \cup B$
    end if
  end for
  $F := F \cup C'$
end for

The next $\mathcal{T}$-Solver is an abstract algorithm for any theory solvers. The algorithm detail is different depended on the theories.

Algorithm 9 $\mathcal{T}$-Solver(a set of boolean values or formulas Δ)

INPUT: A conjunctive boolean values or formulas Δ
OUTPUT: A conjunctive boolean values or formulas Δ', if given Δ conflicts, Δ' is backtracked from Δ due to the conflict.
A conflict clauses Γ, where $\Gamma \subseteq \Delta$.

while $l \in \Delta$ **do**
  if l is a formula **then**
    $x :=$ is the solution of l by the decision procedure of the theory $\mathcal{T}$
    for all $\delta \subseteq \Delta \wedge l \notin \delta$ **do**
      if x conflict with δ **then**
        $\Gamma \subseteq (l \cup \delta)$ ▷ To find conflict clauses Γ is a blackbox depended on the theory
        $S :=$ is the second from the last of Γ
        $s :=$ is the index of S in Δ
        $\Delta' :=$ first s items of Δ ▷ remove $l_{s+1} \dots l_n$ in Δ
        return Δ', Γ
      end if
    end for
  end if
end while
return $\Delta, \Gamma = \emptyset$

Finally, we define $\mathcal{T}$-DPLL algorithm as a very lazy variation DPLL(T).

Algorithm 10 $\mathcal{T}$-DPLL(a $\mathcal{T}$-formula φ)

INPUT: A $\mathcal{T}$-formula φ
OUTPUT: SAT or UNSAT. If SAT, also returns satisfiable assignments

```
F = φ
D = ∅                                    ▷ decided boolean literals
L = ∅                                    ▷ L is learnt clauses from conflict

while true do
  Ψ := T2B(F ∪ L)                        ▷ Boolean abstraction
  status, D := DPLL(Ψ, D)                ▷ Calls DPLL SAT solver
  if status = UNSAT then
    return UNSAT
  else
    Δ := B2T(D)
    Δ′, Γ = 𝒯-Solver(Δ)                  ▷ Theory level check, backtrack and clause learning
    if Γ = ∅ then
      return SAT, Δ′                     ▷ If conflict clauses are empty, it means SAT
    else
      D := T2B(Δ′)                       ▷ Δ′ are backtracked formulas
      L := L ∪ T2B(¬Γ)                   ▷ Γ are conflict clauses
    end if
  end if
end while
```

Chapter 5

Cylindrical Algebraic Decomposition

Cylindrical Algebraic Decomposition (CAD) is an efficient algorithm for Nonlinear Real Arithmetic as we see in the introduction.

In this section, we see the algorithms of Cylindrical Algebraic Decomposition.

The general reference of this section is chapter 1, 2 in [27], chapter 3, 4 in [21], chapter 3 in [19].

5.1 Quantifier Elimination

Quantifier Elimination is a way to get the equivalent quantifier free formula from quantified formula.

The crucial points of Quantifier Elimination in Nonlinear real arithmetic are **Talski-Seidenburg** theorem and **Tomm's lemma**.

First, we see Talski-Seidenberg Theorem.

Theorem 5.1 (Talski-Seidenberg Theorem)**.** *Given a system of polynomial equalities and inequalities in the variables* $x_1, \ldots, x_n$ *and* x_{n+1} *in* $\mathbb{R}$ *with coefficients in* $\mathbb{Z}$

$$\mathcal{P} = \left\{ \begin{array}{lll} P_1(x_1, \ldots, x_n, x_{n+1}) & \triangleright_1 & 0 \\ P_2(x_1, \ldots, x_n, x_{n+1}) & \triangleright_2 & 0 \\ \vdots & & \\ P_m(x_1, \ldots, x_n, x_{n+1}) & \triangleright_m & 0 \end{array} \right\}$$

where the $\triangle_m$ *are* $=$ *or* $\neq$ *or* $>$ *or* $\geq$*, produces a finite set* $Q_1(x_1, \ldots, x_n)$*, ...,* $Q_k(x_1, \ldots, x_n)$ *of systems of polynomial equations and inequalities in* $x_1, \ldots, x_n$ *with coefficients in* $\mathbb{Z}$ *such that, for every* $\alpha \in \mathbb{R}^n$*, the system* $\mathcal{P}$ *has a real solution if and only if one of the* $Q_1(\alpha) \ldots Q_k(\alpha)$ *is satisfied.*

This theorem says $\exists x_{n+1}(\mathcal{P}(x_1, \ldots, x_n, x_{n+1}))$ is equivalent to the disjunction $\bigvee_{i=1}^{i=k}(Q_i(x_1, \ldots, x_n))$. Meaning that there is an algorithm for eliminating the real variable x.

Proof. Section 1.3 in [27] □

Next, we see Thom's lemma.

Lemma 5.2. *(Thom's lemma) The conjunction of the inverse image of the sign of the derivatives of degree* m *mono-variant polynomial* F *such that* $\{x \in R \mid \bigwedge_{f \in F} \mathsf{sign}(f(x)) = \varepsilon(f)\}$*, where* $\varepsilon(f)$ *is a sign condition in* $\{-, +, 0\}$*,* $F = \{f, f', \ldots, f^{(m)}\}$*, is either a point, an open interval, or the empty set.*

This lemma says the regions that consist of the roots and between the roots are definable with the conjunction of the sign conditions.

An algorithm for the theorem and the lemma is **Cylindrical Algebraic Decomposition** we see in the following sections.

5.2 Cylindrical Algebraic Decomposition

Cylindrical Algebraic Decomposition creates the sign invariant regions called Cells from the given systems of polynomial equations and inequalities $\mathcal{P}$. The CAD algorithms is divided into the following three algorithms:

Projection: Projection algorithm does mapping from $n+1$-variant $\mathbb{R}^{n+1}$ space to n-variant $\mathbb{R}^n$ space with the fact that the sign is changed only when the number of roots is changed. Projection uses sub-resultant coefficients to know the degrees of several GCDs so that we can make sure the number of common roots and the number of roots without multiplicity.

Base: Base algorithm captures, and expresses where is the roots with sturm algorithm. Furthermore, base algorithm decompose mono variant $\mathbb{R}$ space into sign invariant regions with the roots.

Lift: Lift algorithm does backward mapping from n to $n+1$ assigning the sample value that is initially picked-up from the sign invariant regions made by base algorithm. Lift algorithm also calculates the minimal polynomial with primitive element algorithm when the sample point is an algebraic number, not a rational. At last, lift algorithm decomposes n-variant $\mathbb{R}^n$ space into sign invariant regions employing the base algorithm over the minimal polynomial.

5.3 Definition and Notation

In this section, we give the definition and the notation for Cylindrical Algebraic Decomposition.

The definitions here follows [28] and [29].

Definition 5.3 (Semi-algebraic set)**.** A set is a **semi-algebraic set** if it is constructed by finitely many applications of union, intersection and complement operation on sets of the form

$$\{x \in \mathbb{R}^n \mid f(x) \geq 0\},$$

where $f \in \mathbb{R}[x_1, \ldots, x_n]$.

Example 5.1. $(-x-6>0) \wedge (x^4-4<0)$*,* $(x=0) \vee (x^2-8<0)$ *are semi algebraic set.*

Definition 5.4. The function $\mathrm{sign} : R \to \{-1, 0, 1\}$ is defined by

$$\mathrm{sign}(r) = \begin{cases} 1 & \text{if} \quad r > 0, \\ 0 & \text{if} \quad r = 0, \\ -1 & \text{if} \quad r < 0. \end{cases}$$

Definition 5.5. Let $\mathcal{C} \subset \mathbb{R}^n$ and $f \in K[x_1, \ldots, x_n]$. Then f is **sign-invariant** on $\mathcal{C}$, if $\forall \alpha, \beta \in \mathcal{C}$ $\mathrm{sign}(f(\alpha)) = \mathrm{sign}(f(\beta))$. Given a set $\mathcal{F} = \{f_1, \ldots, f_m)$ where each element is $f \in K[x_1, \ldots, x_n]$, we say the set is $\mathcal{F}$**-sign-invariant** if all f is sign-invariant on $\mathcal{C}$.

Definition 5.6. A **region** R is a connected subset of $\mathbb{R}^n$.

Definition 5.7. Given a region R, **cylinder over** R, written $Z(R)$ is the set,

$$Z(R) = R \times \mathbb{R}^1 = \{(\alpha, x) \mid \alpha \in R, x \in \mathbb{R}^1).$$

Definition 5.8. Let f, f_1, f_2 be continuous functions on a region R. A **f-section** of $Z(R)$ is the set

$$\{(\alpha, f(\alpha)) \mid \alpha \in R\}$$

and a **(f_1, f_2)-sector** of $Z(R)$ is the set

$$\{(\alpha, \beta) \mid \alpha \in R, f_1(\alpha) < \beta < f_2(\alpha)\}.$$

Definition 5.9. Given a region R, a **decomposition** of R is a finite disjoint regions C_i whose union is R,

$$R = \bigcup_{i=0}^{k} C_i, C_i \cap C_j = \emptyset, i \neq j.$$

Definition 5.10. Given a region R, a **stack over** R is a decomposition which consists of f_i-sections and (f_i, f_{i+1})-sectors.

Definition 5.11. A decomposition $\mathcal{D}$ of $\mathbb{R}^n$ is **cylindrical** if

$n = 1$, $\mathcal{D}$ is a stack over $\mathbb{R}^1$.

$n > 1$, there is a decomposition $\mathcal{D}' = \bigcup_{i=0}^{k} C_i$ of $\mathbb{R}^{n-1}$ such that for each region C_i, there is a subset of $\mathcal{D}$ which is a stack over C_i.

Definition 5.12. A **Cylindrical Algebraic Decomposition** (CAD) of $\mathbb{R}^n$ is a decomposition which is cylindrical and all its component is a semi-algebraic set.

Definition 5.13. Each component that is made by Cylindrical Algebraic Decomposition(CAD) is also called CAD, or **Cell**.

5.4 Projection

We introduce a definition **delineable** which is play an essential role in CAD Projection.

Definition 5.14. We say $\mathcal{F} = f_1, \ldots, f_r \subset \mathbb{Q}[x_1, \ldots, x_n]$ is **delineable** on C' if

1. The total number of complex roots of f_i is remains invariant,
2. the total number of distinct complex roots of f_i is remains invariant,
3. the total number of common complex roots of f_i and f_j is remains invariant (counting multiplicity).

The crucial idea of projection phase is finding a regions where the given polynomials has the constant number of real roots. To find such a region, we count the complex roots in delineable definition.

Because the only time to change the number of real roots is, the pair of complex conjugate roots becomes real roots, or the real root becomes the pair of complex conjugate roots.

To transition from the pair of complex conjugate to a real root, it must go through a real double root. The opposite transition is also must once become a real double root. Thus if the number of complex roots and distinct complex roots remains invariant, the both transition: complex to real, real to complex must not be occur. Therefore the number of real roots is remains invariant in the definition of delineable.

Projection Algorithm

With the delineable definition, and j-th sub-resultant coefficient (3.11) with the theorem of the degree GCD (3.13), we give Projection algorithm.

To define the algorithm we give a function T_k which takes until the k-th degree terms from the polynomial. $T_k(f) = c_k x^k + \cdots + c_0$ where f is $k \le n$ degree polynomial such that $f = c_n x^n + \cdots + c_1 x + c_0$. Why we need this function is because the degree of polynomial depends on the given value of the variant, for example $f(0, y) = 3x^2 + xy^2 + y = y$. Then $\deg_y(f) = 2$, but $\deg_y(f(0, y)) = 1$.

Algorithm 11 Projection$_1(\mathcal{F}_n)$

INPUT: n-variant m polynomials $\mathcal{F}_n$ $\quad \triangleright\ \mathcal{F}_n = F_1(x_1, \ldots, x_n), \ldots, F_m(x_1, \ldots, x_n)$
OUTPUT: a finite set of $n-1$ variant polynomials

$\alpha := \{x_1, \ldots, x_{n-1}\}$
while $f(x_1, \ldots, x_{n-1}, x_n) \in \mathcal{F}$ **do**
  $\mathcal{F}' = \emptyset$
  $k := \deg(\mathrm{COEFF}_{x_n}(f))$
  if $k \ne 0$ **then**
    $\mathcal{F}' := \mathcal{F}' \cup \mathrm{COEFF}_{x_n}(f)$
  end if
  for all $0 \le l \le k$, where $k = \deg_{x_n}(f(\alpha))$ **do**
    if $\mathrm{PSC}_l(T_k(f), T_k(f'))$ $\ne$ constant value **then**
      $\mathcal{F}' := \mathcal{F}' \cup \mathrm{PSC}_l(T_k(f), T_k(f'))$
    end if
  end for
end while
for all $0 \le i < j \le m$, $0 \le l < k$, $f_i, f_j \in \mathcal{F}$, where $k = \min\{\deg_{x_n}(f_i), \deg_{x_n}(f_j)\}$ **do**
  $k_i := \deg_{x_n}(f_i(\alpha))$
  $k_j := \deg_{x_n}(f_j(\alpha))$
  if $\mathrm{PSC}_l(T_{k_i}(f_i), T_{k_j}(f_j))$ $\ne$ constant value **then**
    $\mathcal{F}' := \mathcal{F}' \cup \mathrm{PSC}_k(T_{k_j}(f_i), T_{k_j}(f_j))$
  end if
end for
return $\mathcal{F}'$

Since Projection_1 erase one variable from the polynomials, we recursively apply Projection_1 until it is mono variant polynomials.

Algorithm 12 Projection($\mathcal{F}_n$)

INPUT: n-variant m polynomials $\mathcal{F}_n$
OUTPUT: a finite set of $1, \ldots, n$-variant polynomials $\mathcal{F}_1, \ldots, \mathcal{F}_n$
for $i = n$ **to** $i = 2$ **do**
  $\mathcal{F}_{i-1} := \text{Projection}_1(\mathcal{F}_i)$
end for
return $\mathcal{F}_1, \ldots, \mathcal{F}_n$

Example 5.2. *Given* $\mathcal{F}_2 = \{f_1 = -x^2+y^3+3y^2-2, f_2 = x^2+y^2+6y+1, f_3 = xy-x-6\}$, *Projection*$_1(\mathcal{F}_2)$ *be the following.*

First, the $\mathsf{COEFF}_y(f_i)$ *be* $\{-x^2-2,\ x^2+1,\ x,\ -x-6\}$.

In f_1 *and* f_2, y*'s coefficient is always* 1 *so for any* y *the degree is* 2. *However in* f_3, *the coefficient of* y *is* x, *thus we need to calculate* $T_k(f_3)$, $T_0(f_3) = -x-6, T_1(f_3) = xy-x-6$.

Next, calculating $\mathsf{PSC}_k(f_i, f_i')$ *and* $\mathsf{PSC}_k(f_i, f_j)$, *we get,* $\mathsf{PSC}_0(f, f') = 27x^4-108$, $\mathsf{PSC}_0(f_2, f_2) = 4x^2-32$, $\mathsf{PSC}_0(f_1, T_1(f_3)) = x^5-2x^3-54x^2-216x-216$, $\mathsf{PSC}_0(f_1, T_0(f_3)) = -x^3-18x^2-108x-216$, $\mathsf{PSC}_0(f_2, T_1(f_3)) = x^4+8x^2+48x+36$, $\mathsf{PSC}_0(f_2, T_0(f_3)) = x^2+12x+36$. *Thus omitting the constant multiples, we get*
Projection$_1(\mathcal{F}_2) = \mathcal{F}_1 = \{-x^2-2, x^2+1, x, -x-6, x^4-4, x^2-8, x^6-17x^4+61x^2+188, x^5-2x^3-54x^2-216x-216, -x^3-18x^2-108x-216, x^4+8x^2+48x+36\}$

5.5 Base

5.5.1 Sturm's theorem

Definition 5.15. (Sturm sequence) Given a f, let $f_1 = f, f_2 = f'$, we calculate a variation of euclidean algorithm whose difference is the sign of the reminder,

$$f_1 = q_1 f_1 - f3$$
$$f_2 = q_2 f_3 - f4$$
$$\vdots$$
$$f_{k-1} = f_{k-1} f_k + 0,$$

where $f_k = \mathsf{GCD}(f, f_1)$. This is called a **strum sequence** of f.

Example 5.3. *Let* $f = x^5 - 2x^3 - 54x^2 - 216x - 216$*, the sturm sequence* f_i *be* $f_1 = x^5 - 2x^3 - 54x^2 - 216x - 216$*,* $f_2 = 5x^4 - 6x^2 - 108x - 216$*,* $f_3 = 4x^3 + 162x^2 + 864x + 1080$*,* $f_4 = -\frac{28461}{4}x^2 - 42282x - 54459, f_5 = -215267040x - 396154368$*.*

Definition 5.16. We writes the number of sign variation f with sturm sequence on $x \in R$ which denote $\mathbf{V_f(x)}$.

Example 5.4. *Given* $f = x^5 - 2x^3 - 54x^2 - 216x - 216$*, the values on* $x = 10$ *are* $\{90224, 48104, 5984, -1188804, -\frac{2548824768}{90003169}, \frac{416110841209179}{558688977025}\}$*, the signs are* $\{+, +, +, -, -, +\}$*, thus the number of sign variations* $V_f(10) = 2$*.*

Theorem 5.17 (Sturm's Theorem)**.** *Let* $f(x)$ *be a mono variant polynomial, a, b* $\in \mathbb{R}$ *where* $a < b$*. The number of roots of f in the interval* (a, b) *is equal to* $V_(a) - V_(b)$*.*

Proof. See §15 in [19], or section 1.1 in [27]. □

5.5.2 Root finding algorithm

We give the algorithm to capture where is the real roots with sturm sequence. To start using strum sequence, we need a sane bound of the roots of a polynomial.

Proposition 5.18 (A root bound)**.** *Let* $f = a_0x^m + a_1x^{m-1} + \cdots + a_m$*, where* $a_0 \neq 0$*. If* r *is a root of* f*, set* $M = \max_{i=1,\dots,m}(a_i)$*, then*

$$|r| \leq M + 1 \tag{5.1}$$

and denotes $M + 1$ *be* $\mathrm{bound}(f)$*.*

Proof. See §19 in [19]. □

This root bound is not the best, however we use this here because it is easy to understand.

Using the root bound, we give RootInterval algorithm to get the intervals where are the real roots.

Algorithm 13 RootInterval(f)

INPUT: a mono variant polynomial f
OUTPUT: the real roots intervals $[a_0, b_0] \ldots [a_n, b_n]$

$a := -bound(f)$
$b := bound(f)$
$e := b$
$r := \emptyset$

if $V_f(a) - V_f(e) = 0$ **then**
  return $\emptyset$
end if

while $a \neq e$ **do**
  while $V_f(a) - V_f(b) > 1$ **do**
    $b = b/2$
  end while
  $r := [a, b] + r$
  $a := b$
  $b := e$
end while

Given a mono variant polynomial f, this root interval algorithm returns the all real root intervals.

5.5.3 Base Algorithm

With the RootInterval algorithm by strum sequence and the square free polynomial algorithm by GCD, we give Base algorithm.

Algorithm 14 Base($\mathcal{F}_1$)

INPUT: a finite set of mono variant polynomials $\mathcal{F}_1$
OUTPUT: the real roots and the intervals $\mathcal{R}$
$\Pi(f) :=$ SQfree($\prod_{f \in \mathcal{F}_1}(f)$) ▷ get the square free polynomial to get different roots
$\mathcal{R} := \emptyset$
while $f \in \Pi(f)$ **do**
  if $deg(f) = 1$ **then**
    $\mathcal{R} := \mathcal{R} \cup$ the solution of f ▷ if $f = x + 1$, $x = -1$
  else
    $\mathcal{I} :=$ RootInterval(f)
    for all $I \in \mathcal{I}$ **do**
      $\mathcal{R} := \mathcal{R} \cup \{f, I\}$ ▷ add the polynomial and the interval
    end for
  end if
end while
return $\mathcal{R}$

In this algorithm we calculate the square free polynomial after we make the product of polynomials. We keep the polynomials inside the production. For example if the square free polynomial is $\Pi(f) = (x-1)(x^2+2)$, we keep the formula, we do not calculate $(x-1)(x^2+2) = x^3 - x^2 + 2x - 2$ so that we can get the linear solutions.

Example 5.5. *Given* $\mathcal{F}_1 = \{f_1 = -x-6, f_2 = x^5 - 2x^3 - 54x^2 - 216x - 216,\}$*, Base(*$\mathcal{F}_1$*)* $=$ $\mathcal{R}^1$ *is* $\{-6, \{f_1 = x^5 - 2x^3 - 54x^2 - 216x - 216, I = [4.8, 4.9]\}\}$.

5.6 Lift

In this section, we give Lift algorithm. Inside the algorithm, we use sample points.

Definition 5.19. Given real roots with the intervals, **sample points** are the real points and the intermediate point between the roots.

Example 5.6. *If given real roots with the intervals* $\{0, \{[x^2-2], -1, 5-1.4\}, 1, \{x^2-2, [1.4, 1.5]\}, 4\}$*, a sample points are* $\{-1, 0, -1, \{x^2-2, [-1, 5-1.4]\}, 0, \{[x^2-2], 1.4, 1.5\}, 2, 4, 5\}$.

With sample points, we define Lift algorithm.

Algorithm 15 Lift$_1$($\mathcal{F}_{n+1}$, $\mathcal{R}^n$)

INPUT: a finite set of $n+1$-variant polynomials $\mathcal{F}_{n+1}$,
and the real roots $\mathcal{R}^n$ for n variant polynomials $\mathcal{F}_n$
OUTPUT: the real roots $\mathcal{R} \times \mathcal{R}^n$ for $n+1$-variant polynomials $\mathcal{F}_{n+1}$

$\mathcal{S}_n :=$ sample points for the real roots $\mathcal{R}_n$
$\mathcal{F}_1 = \emptyset$ ▷ mono variant polynomials
for all $f \in \mathcal{F}_{n+1}$ **do**
  for all $S \in \mathcal{S}_n$ **do**
    for all $s \in S$ if s is a real number for x_i **do**
      $f := f[x_i/s]$ ▷ replace the variable with the real number s
    end for
    for all $p \in S$ if p is a polynomial for x_j **do**
      $f :=$ MinPolByPrimElem(f, p) ▷ ie $f = f(x, x_1)$, $p = p(x)$
    end for
    $\mathcal{F}_1 := \mathcal{F}_1 \cup f$
  end for
end for
$\mathcal{R} :=$ Base$(\mathcal{F}_1)$
return $\mathcal{R} \times \mathcal{R}^n$ ▷ Cartesian product

Given a finite set of 2-variant polynomials $\mathcal{F}_2$, and the real roots $\mathcal{R}^1$ for mono variant polynomials $\mathcal{F}_1$, this Lift$_1$ calculates $\mathcal{R}^2$ real roots for 2 variant polynomials $\mathcal{F}_2$. Meaning that it lift up the mono variant polynomial to n-variant polynomials.

We define Lift algorithm recursively using Lift_1. It calculate the real roots of n-variant polynomials $\mathcal{F}_n$.

Algorithm 16 Lift($\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, $\mathcal{R}^1$)

INPUT: a finite family of n-variant polynomials set $\mathcal{F}_1, \ldots, \mathcal{F}_n$,
and the real roots $\mathcal{R}^1$ for mono variant polynomials $\mathcal{F}_1$
OUTPUT: the real roots $\mathcal{R}^2, \ldots, \mathcal{R}^n$ for 2 to n-variant polynomials $\mathcal{F}_2, \ldots, \mathcal{F}_n$

for $i = 2$ **to** $i = n$ **do**
  $\mathcal{R}^i := \text{Lift}_1(\mathcal{F}_i, \mathcal{R}^{i-1})$
end for
return $\mathcal{R}^2, \ldots, \mathcal{R}^n$

Example 5.7. *Let* $\mathcal{F}_2 = \{F_1 = -xy - x - 6\}$, $\mathcal{R}^1 = \{-6, \{f_1 = x^5 - 2x^3 - 54x^2 - 216x - 216, I = [4.8, 4.9]\}\}$. *Since* $F_1(-6) = -6y$, *the root is* $y = 0$. *For* $x = \{f_1 = x^5 - 2x^3 - 54x^2 - 216x - 216, I = [4.8, 4.9]\}$, F_1 *be* $y^5 + y^4 - 5y^3 + y^2 + 4y - 38$ *by MinPolByPrimElem, and the root be* $y = \{y^5 + y^4 - 5y^3 + y^2 + 4y - 38, I = [2.1, 2.3]\}$. *Thus* $\text{Lift}_1(\mathcal{F}_1, \mathcal{R}^1) = \mathcal{R}^2 = \{0, \{y^5 + y^4 - 5y^3 + y^2 + 4y - 38, I = [2.1, 2.3]\}\}$

5.7 Quantifier Elimination by Cylindrical Algebraic Decomposition

5.7.1 QE-CAD Algorithm

We give the algorithm doing Quantifier Elimination by Cylindrical Algebraic Decomposition: named QE-CAD algorithm. To define the algorithm, we make two sub algorithm **Sign**, and **ThomsEncoding**. Sign algorithm determines the sign of the given polynomial at the point where includes algebraic numbers. ThomsEncoding defines the Cell which is given by a sample point, using Thom's lemma (5.2).

Algorithm 17 Sign(f, α)

INPUT: a n-variant polynomial f, and
the point $\alpha = (r_1, \ldots, r_n)$ in the sample points $\mathcal{S}^n$ that specify a cell
OUTPUT: the sign of the polynomial f at the point α

if all $r \in \alpha$ are real numbers **then**
  return sign($f(\alpha)$) ▷ assign the real numbers, and get the sign with sign function
end if
for $i = 1$ **to** $= n$ **do**
  if $r_i \in \alpha \wedge r_i$ is a real number **then**
    $f = f[x_i/r_i]$ ▷ replace the variable with the real number
  end if
end for
for all $r \in \alpha \wedge$ r is an algebraic number **do**
  f' := the defined polynomial of r
  $f = f/f'$ ▷ f divide by f'
end for
if $f = 0$ **then**
  return 0 ▷ if it is divided, the sign is 0
else
  for all $r_i \in \alpha$, r_i is an algebraic number **do**
    I := get the interval of r_i
    m := get a point inside the interval I
  end for
  $f = f[x_i/r_i]$ ▷ since the point is not 0, so the sign does not change in the interval
end if
return sign(f)

Algorithm 18 ThomsEncoding($\mathcal{F}_n$, α)

INPUT: a set of n-variant polynomials $\mathcal{F}_n$, and
the point $\alpha = (r_1, \ldots, r_n)$ in the sample points $\mathcal{S}^n$ that specify a cell
OUTPUT: The definition of the cell expressed by the semi-algebraic set

$\Pi(F) :=$ SQFree($\prod_{f \in \mathcal{F}}(f)$)
$\mathcal{T} := \emptyset$

for $i = 0$ **to** $i = deg(\Pi(F))$ **do**
  $f^{(i)} :=$ i-th derivative of $\Pi(F)$ ▷ 0-th derivative is f itself $f^{(0)} = f$
  $\sigma_i :=$ Sign($f^{(i)}, \alpha$)
  ▷ := set either ">","=" or "<" by the sign σ_i
  $T := f^{(i)} \triangleright 0$
  $\mathcal{T} := \cup\, T$
end for
return $\bigwedge_{T \in \mathcal{T}}(T)$ ▷ returns the conjunction $f \triangleright_1 0 \wedge f^{(1)} \triangleright_2 0 \wedge \ldots \wedge f^{(n)} \triangleright_n 0$

Algorithm 19 QE-CAD($\mathcal{P}$)

INPUT: a system of m polynomial equalities or inequalities $\mathcal{P}$
OUTPUT: a set of cells such that the cells satisfy the $\mathcal{P}$

$\mathcal{F}_n :=$ the polynomials of $\mathcal{P}$
$\mathcal{F}_1, \ldots, \mathcal{F}_n :=$ Projection($\mathcal{F}_n$)
$\mathcal{R}^1 :=$ Base($\mathcal{F}_1$)
$\mathcal{R}^2, \ldots, \mathcal{R}^n :=$ Lift($\{\mathcal{F}_1, \ldots, \mathcal{F}_n\}, \mathcal{R}^1$)
$\mathcal{S}^2 :=$ the sample points of $\mathcal{R}^n$

$\mathcal{C} = \emptyset$
for all $\alpha = (r_1, r_2, \ldots, r_n) \in \mathcal{S}^n$ **do**
  for all $i = 1$ **to** $i = m$, Sign(f, α) satisfies $P_i \in \mathcal{P}$, where f is a polynomial of P_i **do**
    $T :=$ ThomsEncoding($\mathcal{F}_n, \alpha$)
    $\mathcal{C} := \mathcal{C} \cup T$
  end for
end for
return $\mathcal{C}$

5.7.2 Example of QE-CAD

In this section, we see how QE-CAD works by an example.

Let $\mathcal{P}(x, y) = \{-x^2 + y^3 + 3y^2 - 2 < 0, x^2 + y^2 + 6y + 1 < 0, xy - x - 6 > 0\}$, and the quantified formula be $\exists y(\mathcal{P}(x, y))$.

QE-CAD($\exists y(\mathcal{P}(x, y))$) is calculated by the following procedure,

$\mathcal{F}_2 = \{-x^2 + y^3 + 3y^2 - 2, x^2 + y^2 + 6y + 1, x, xy - x - 6 > 0\}$.

$$
\begin{aligned}
\mathtt{Projection}(\mathcal{F}_2) &= \mathcal{F}_1 \\
&= \{-x - 6, x^2 - 8, x^4 + 8x^2 + 48x + 36, \\
&\quad x, x^4 - 4, x^5 - 2x^3 - 54x^2 - 216x - 216\} \\
\mathtt{Base}(\mathcal{F}_1) &= \mathcal{R}^1 \\
&= \{-6, \\
&\quad \{x^2 - 8, [-2.9, -2.8]\}, \\
&\quad \{x^4 + 8x^2 + 48x + 36, [-2.4, -2.3]\}, \\
&\quad \{x^4 - 4, [-1.5, -1.4]\}, \\
&\quad \{x^4 + 8x^2 + 48x + 36, I = [-0.9, -0.8]\}, \\
&\quad 0, \\
&\quad \{x^4 - 4, [1.4, -1.5]\}, \\
&\quad \{x^2 - 8, [2.8, 2.9]\}, \\
&\quad \{x^5 - 2x^3 - 54x^2 - 216x - 216, [4.8, 4.9]\}\}
\end{aligned}
$$

From $\mathcal{R}^1$, we get 19 cells $\mathcal{C}_1, \ldots, \mathcal{C}_{19}$. Checking the sign of the sample points, we get the following 5 cells that satisfy $\mathcal{P}$.

$$
\begin{aligned}
\mathcal{C}_5 &= \{x \mid \{f = x^2 - 8, [-2.9, -2.8]\} < x < \{x^4 + 8x^2 + 48x + 36, [-2.4, -2.3]\}\} \\
\mathcal{C}_6 &= x = \{x^4 + 8x^2 + 48x + 36, [-2.4, -2.3]\} \\
\mathcal{C}_7 &= \{x \mid \{x^4 + 8x^2 + 48x + 36, [-2.4, -2.3]\} < x < \{x^4 - 4, [-1.5, -1.4]\}\} \\
\mathcal{C}_8 &= x = \{x^4 - 4, [-1.5, -1.4]\} \\
\mathcal{C}_9 &= \{x \mid \{x^4 - 4, [-1.5, -1.4]\} < x < \{x^4 + 8x^2 + 48x + 36, [-0.9, -0.8]\}\}
\end{aligned}
$$

Next, for ThomsEncoding, we check the signs of polynomials in $\mathcal{F}_1$ above the cells.

	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$	$\mathcal{C}_9$
$-x-6$	-	-	-	-	-
x^2-8	-	-	-	-	-
$x^4+8x^2+48x+36$	+	0	-	-	-
x	-	-	-	-	-
x^4-4	+	+	+	0	-
$x^5-2x^3-54x^2-216x-216$	-	-	-	-	-

In this example, the signs of polynomials $f \in \mathcal{F}_2$ isolate the cells, thus ThomsEncoding does not require to calculate the derivatives. Each cell is defined by the conjunctions of the signs as a semi-algebraic set.

$$\begin{aligned}
\mathcal{C}_5 =& (-x-6<0) \wedge (x^2-8<0) \wedge (x^4+8x^2+48x+36>0) \wedge (x<0) \wedge (x^4-4>0) \wedge \\
& (x^5-2x^3-54x^2-216x-216<0) \\
\mathcal{C}_6 =& (-x-6<0) \wedge (x^2-8<0) \wedge (x^4+8x^2+48x+36=0) \wedge (x<0) \wedge (x^4-4>0) \wedge \\
& (x^5-2x^3-54x^2-216x-216<0) \\
\mathcal{C}_7 =& (-x-6<0) \wedge (x^2-8<0) \wedge (x^4+8x^2+48x+36<0) \wedge (x<0) \wedge (x^4-4>0) \wedge \\
& (x^5-2x^3-54x^2-216x-216<0) \\
\mathcal{C}_8 =& (-x-6<0) \wedge (x^2-8<0) \wedge (x^4+8x^2+48x+36<0) \wedge (x<0) \wedge (x^4-4=0) \wedge \\
& (x^5-2x^3-54x^2-216x-216<0) \\
\mathcal{C}_9 =& (-x-6<0) \wedge (x^2-8<0) \wedge (x^4+8x^2+48x+36<0) \wedge (x<0) \wedge (x^4-4<0) \wedge \\
& (x^5-2x^3-54x^2-216x-216<0)
\end{aligned}$$

The quantified elimination result is the disjunction of the cell definitions.

$\exists y(\mathcal{P}(x,y)) = \mathcal{C}_5 \vee \mathcal{C}_6 \vee \mathcal{C}_7 \vee \mathcal{C}_8 \vee \mathcal{C}_9.$

Chapter 6

NLSAT

6.1 Introduction

NLSAT takes on the part of nonlinear real arithmetic inside SMT solver Z3 [30], thus it solves the satisfiability problem of nonlinear real arithmetic.

NLSAT is characterized by the following two features.

1. Projection-Based Explanation and Model-Based Projection
2. Model Construction Satsifiability Calculus

6.1.1 Projection-Based Explanation and Model-Based Projection

Projection-Based Explanation and Model-Based Projection [31, 32]: is the usage of CAD algorithm in NLSAT.

When NLSAT detects a conflict, **projection-based explanation** creates new polynomial literals by CAD algorithm which explains the conflict that is expressed as a CAD cell. In other words, the conflict is a cell where is not satisfiable in the current trail. To do this, NLSAT defines **explain** function.

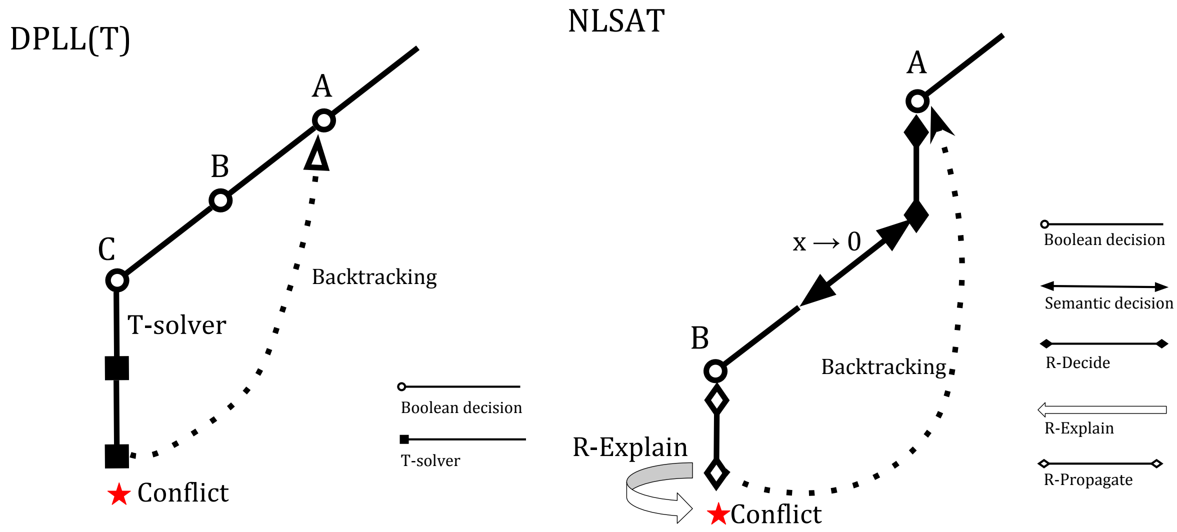
Model-based projection is a specialized CAD in the explain function. It focus only a single cell (region), and calculates only the cell by CAD algorithms using the model in the trail. Why it is called model-based because NLSAT trail is relaxed to store boolean decisions and **semantic decisions**. A semantic decision is a real value of a semantic variable, for example $x = \frac{1}{2}$.

Remark 6.1. In [31, 32], they are named "projection"-based explanation, and model-based "projection". However, NLSAT employs projection, base and lift, and they are model-based.

6.1.2 Model Construction Satsifiability Calculus

Different than other SMT solvers, NLSAT and its successor MCSat [33, 34] handle the both boolean level clauses and the semantic decisions in the special **trail**. This approach is an on-demand approach in SMT solver: **Splitting on-demand** [35]. In DPLL(T) solver, background theory calculation is everything inside $\mathcal{T}$-solver. In contrast, splitting on-demand delegates the internal case splitting inside $\mathcal{T}$-solver into DPLL engine.

With this approach, NLSAT integrate background theory algorithms into DPLL and CDCL [33]. Thus the DPLL algorithms: unit-resolution (unit propagation), conflict-driven clause learning, non-chlonological backtracking employ background theories in NLSAT.

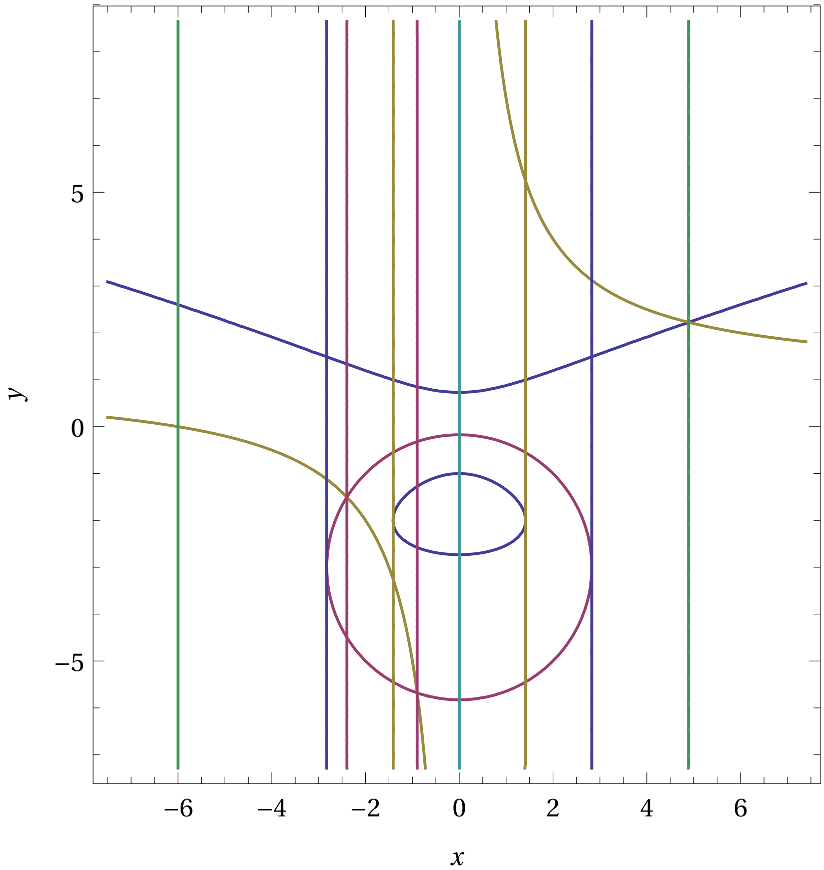


The left figure is the standard DPLL(T) algorithm. The right is the NLSAT. NLSAT decide semantic decisions with R-Decide which uses the result of the base algorithm to know where is the roots. NLSAT unit-resolution algorithm (named R-Propagate) founds a semantic level conflict by R-conflict, again using the result of the base. Then, NLSAT triggers R-Explain with CAD algorithm to explain the conflict, learning new literal (polynomial literal), and doing non-chronological backtracking includes the semantic decisions.

6.2 NLSAT Example

We see how NLSAT(φ) works through an example. The example problem is
$\varphi = (-x^2 + y^3 + 3y^3 - 2 < 0) \wedge (x^2 + y^2 + 6y + 1 < 0) \wedge (xy - x - 6 > 0)$.

We first see the figure by QE-CAD algorithm to compare the difference with NLSAT algorithm. We need to calculate 19 cells for variable x.



In the next page, we see the NLSAT(φ) calculation procedure step by step.

$$M = [\![]\!] \qquad \text{(initial trail)}$$

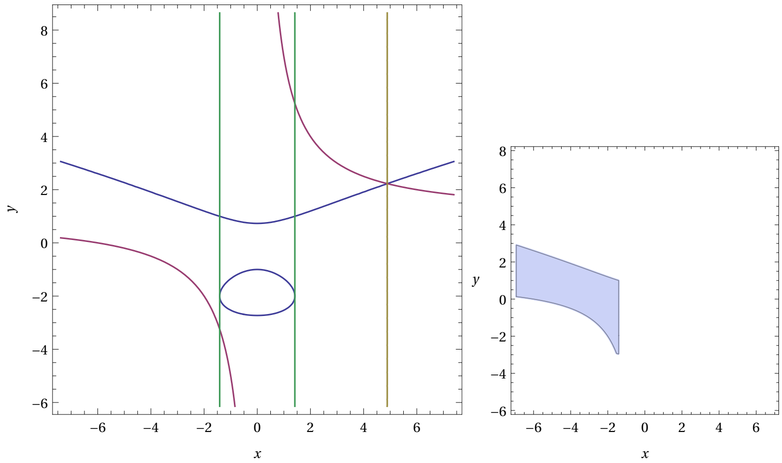
$$M = [\![\ x \overset{1}{\mapsto} -2\ ]\!] \qquad \triangleright \text{ R-decide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0)\ ]\!] \qquad \triangleright \text{ B-decide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ y \overset{3}{\mapsto} 0\ ]\!] \qquad \triangleright \text{ R-decide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ y \overset{3}{\mapsto} 0,\ \neg(xy - x \overset{3}{-} 6 > 0)_{\downarrow E}\ ]\!]$$

$$\begin{aligned} E &= \text{R-Explain}(M) \\ &= (x^4 - 4 > 0 \land 4x^3 < 0 \land \\ &\quad x^5 - 2x^3 - 54x^2 - 216x - 216 > 0) \land \\ &\quad (-x^2 + y^3 + 3y^3 - 2 < 0 \land xy - x - 6 < 0) \end{aligned}$$



$\triangleright$ R-Propagate

(Here the M is B-Conflict because of $\neg(xy - x \overset{3}{-} 6 > 0)_{\downarrow E}$) $\triangleright$ AnalyzeConflict

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ \neg(x^4 \overset{3}{-} 4 > 0)\ ]\!] \qquad \triangleright \text{ BacktrackAndDecide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ \neg(x^4 \overset{3}{-} 4 > 0),\ y \overset{4}{\mapsto} -2\ ]\!] \qquad \triangleright \text{ R-decide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ \neg(x^4 \overset{3}{-} 4 > 0),\ y \overset{4}{\mapsto} -2,\ -x^2 + y^2 \overset{5}{+} 6y + 1 < 0\ ]\!] \qquad \triangleright \text{ B-Decide}$$

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ \neg(x^4 \overset{3}{-} 4 > 0),\ y \overset{4}{\mapsto} -2,\ -x^2 + y^2 \overset{5}{+} 6y + 1 < 0,\ xy - x \overset{6}{-} 6 > 0\ ]\!] \qquad \triangleright \text{ B-Decide}$$

$$\text{SAT},\ M\ (x = -2,\ y = -2) \qquad \triangleright \text{ SAT}$$

In the above example, the explain function R-Explain calculates the single cell which is more efficient to calculate the full CAD. Furthermore, with the result of R-Explain, we get a stronger constraint $\neg(x^4 - 4 > 0)$. Using it, NLSAT find a solution efficiently.

6.3 Discussion on the NLSAT implementation strategy

In this section, we discuss the NLSAT decision of the implementation.

We see these topics:

- Variable selection
- Selection for the value of a semantic decision
- Eager or lazy in NLSAT
- The algorithm of Explain

Variable selection

The variable ordering largely effects in CAD calculation. Thus how we pick-up the variable in R-decide is import.

We denote the number of variable occurrence in the polynomial $\#(f, x_i)$. If $f = x^3+2x+y$, $\#(f, x) = 2$, $\#(f, y) = 1$. x is two times, y is one time in f.

Given a polynomial $f(x_0, \dots, x_n)$, NLSAT is reordering the variable as the following:

$x_i \prec x_j \Leftrightarrow (deg_{x_i}(f) > deg_{x_j}) \vee \#(f, x_i) > \#(f, x_j) \vee ((deg_{x_i}(f) = deg_{x_j}(f)) \wedge i < j)$.

This means high degree first, more constrained first. After reordering, it picks the smaller variable first. This is a simple heuristic. So any other selection is considerable.

This behavior is implemented in the function heuristic_reorder() in the source: z3/src/nlsat/nlsat_solver.cpp.

We can evaluate other options to modify the source of the function.

Selection for the value of a semantic decision

How we set the real value of a semantic decision in R-Decide effects again in CAD especially the lifting phase. If the value is an algebraic number it is required to call MinPolByPrimElem [Algorithm 2] which is a heavy procedure.

If it is possible, NLSAT always set the value of a variable x be a **Dyadic Rationals** such that $\mathbb{D} = \{\frac{p}{2^k} \mid p \in \mathbb{Z}, k \in \mathbb{N}\}$. For example, x be $0, 1, -1, \frac{1}{2}, -\frac{1}{2}, 2$.

If it is not possible to set dyadic rational, nlsat set the algebraic number.

To set this variable, NLSAT randomizes to take the k and p, and find a rational until if the interval is $\frac{1}{2^{32}}$.

This randomization and the limit is implemented in the function
interval_set_manager::peek_in_complement in the source:
z3/src/nlsat/nlsat_interval_set.cpp.

We can evaluate other strategies to modify the function.

Eager or lazy in NLST

Since NLSAT employs model construction satsifiability calculus , when we call background theories is more variable than DPLL(T) solvers.

We can evaluate the behavior with changing the “lazy” option of nlsat. By default it is set as the most eager mode.

This is implemented in the function process_arith_clause()
in the source: z3/src/nlsat/nlsat_solver.cpp.

Furthermore, comparing several versions of SMT solver yices [36] with and without MCSat, we can make sure the contribution to the performance of model construction satsifiability calculus.

The algorithm of Explain

As we see in the previous sections, NLSAT uses CAD for the explain function. However it is not mandatory to use CAD for explain. Jovanovic and Moura proposed to use an algorithm which is more efficient, but does not guarantee the termination [31].

Moreover, a singly exponential complexity algorithm that is dedicated for the satisfiability problem of nonlinear arithmetic is proposed in the chapter 13 in [22]. To the best of my knowledge, there is no SMT solver which implements this algorithm now in August 2016.

The NLSAT explain is implemented in the source: z3/src/nlsat/nlsat_explain.cpp.

It is possible to evaluate the choice of explain algorithm rewriting the explain algorithm there.

Chapter 7

How NLSAT works

In this chapter, we give the algorithm of NLSAT to make sure how it works.

7.1 NLSAT trail

All the characteristics of NLSAT (and its successor MCSat) are come from its data structure of **trail**. The technical term trail is found in modern SAT solvers, but NLSAT relaxing the trail data-type, and the purpose. The following definition refers [34].

Definition 7.1 (trail in NLSAT)**.** The trail in NLSAT is a sequence of trail elements: a boolean decision, a semantic decision, a clausal propagation, a semantic propagation.

- A **boolean decision** is a literal L that we assume to be true. This is the same with decided literals in modern SAT/SMT solvers.
- A **semantic decision** is a decision on the value of a non-Boolean variable (ie a variable in a polynomial), it is denoted $x \mapsto \alpha$ where α is the decision value of the variable.
- A **clausal propagation** is a literal L derived to be true through clause C using UnitResolution, which is denoted by $L_{\downarrow C}$. If C is a unit clause $C = L$ it is denoted by $L_{\downarrow}$.
- The **level** of a element in a trail is the number of decision in the trail up to and including the element itself. Since NLSAT is branching both boolean and semantic decision, the level is ie the number of branching. We write the level above each element ie $\overset{level}{L}$

- A literal L is **evaluated** if and only if all the semantic decisions of L is available in the trail, else the literal is **undefined** in the trail.
- A **semantic propagation** is a literal L is evaluated to be $\mathtt{true}$ in a trail, which is denoted by $L_{\downarrow k}$, where the k is the level of the highest semantic decision used in evaluating L.

Modern SAT solvers only adds boolean decision and clausal propagation into the trails. In contrast, NLSAT is branching both in boolean and semantic level variables, and adds both of them into the trail. Furthermore, NLSAT adds new polynomial constraints literals, that is not available in the original $\mathcal{T}$-formula, into the trail as the **boolean decision**. NLSAT/MCSat uses the trail as **Clause and Variable database** [34].

Example 7.1. *Let the $\mathcal{T}$ formula $\varphi = (-x^2 + y^3 + 3y^3 - 2 < 0) \wedge (x^2 + y^2 + 6y + 1 < 0) \wedge (xy - x - 6 > 0)$. Then, a trail is*

$$M = [\![\ x \overset{1}{\mapsto} -2,\ (-x^2 + y^3 \overset{2}{+} 3y^3 - 2 < 0),\ \neg(xy - y \overset{1}{-} 6 > 0)_{\downarrow 1},\ (x^2 + y^2 \overset{2}{+} 6y + 1)_{\downarrow C}\ ]\!].$$

The first element is a semantic decision, the second is a boolean decision, the third is a semantic propagation, the last is a clausal propagation.

7.2 Projection-Based Explanation and Model-Based Projection in NLSAT

For explain, we gives R-Explain algorithm. In the algorithm we use CAD Projection [Algorithm 12], Base [Algorithm 14], Lift [Algorithm 16].

Algorithm 20 R-Explain(M, C)

INPUT:INPUT: A trail M, and
a clause C is conflicted in M
OUTPUT:A clause E as the explanation of the conflict with CAD algorithm

$\mathcal{N}$:= is polynomial constraints in M ▷ This is smaller than the original constrains
$x_1, \ldots, x_n := (v_1, \ldots, v_n)$, where v_i is the semantic decision of x_i in M

$\mathcal{F}_n$:= the polynomials in $\mathcal{N}$
$\mathcal{F}_1, \ldots, \mathcal{F}_n$:= Projection$_e(\mathcal{F}_n)$
$\mathcal{R}^1$:= Base$(\mathcal{F}_1)$ ▷ $\mathcal{R}^1$ is the roots of $\mathcal{F}_1$
$\mathcal{C}^1$:= is the cells by the roots $\mathcal{R}^1$
C^1 := is the cell $C^1 \in \mathcal{C}^1$ where $(x_1 = C^1) \vee (x_1 \in C^1)$
$C^1, \ldots, C^n$:= Lift$_e(\mathcal{F}_1, \ldots, \mathcal{F}_n$, C^1, $\{x_2, \ldots, x_n\})$
$E := \bigwedge_{i=1}^{n}(C_i)$
return E

First, R-Explain only Projection the literals that are next to the point by the specialized projection Projection$_e$. Next, R-Explain get the cell definition where the value $x_1 = v_1$ is included from the result of Base $\mathcal{R}^1$. Similarly, in Projection and Lift, Projection$_e$, Lift$_e$ only calculate the cell where $x_2, \ldots, x_n$ included. Then returns E as the conjunction of the cell definition. E express the Cell in R^n.

Why CAD in NLSAT is efficient?

Why CAD in NLSAT is efficient is because:

- The problem size is smaller than the original problem. R-Explain is targeting only the literals (polynomial constraint) in the trail, thus most of the time it is smaller than the original problem.
- The special Projection, Projection$_e$ targets only the literals containing the point which is specified the semantic decisions. For example, if the literal is $f(x, y) = x^2 + y^2 - 2$, and current semantic decisions are $x \mapsto 0$, $y \mapsto -1$, Projection$_e$ calculates $f(0, y) = y^2 - 2$ and $f(x, -1) = x^2 - 1$, and get the roots of x and y inside Projection$_e$. Similarly, for all polynomial constraint literals in M, Projection$_e$

calculates all the roots, then sorting the roots, finally get what semantic literals are containing the points. Then projection only the literals.

- R-Explain only lift the single cell where the point of semantic decisions in the current trail. It calls the special variation of Lift, Lift_e which Lift only the cell.

Remark 7.2. NLSAT can calculate only open cells called Single Open Cell in [37] if the problem is **Full dimensional**. Full dimensional is a problem where all the constrains are inequalities. Since open cells represented by inequalities, it does not require to calculate primitive element.

7.3 Model Constructing Satisfiability Calculus in NLSAT

To do CDCL for the first-order setting, we gives sub-algorithms for NLSAT. They are Propagate, Decide, AnalyzeConflict, BackTrackAndDecide. Propagate does resolution and checks the conflict, Decide does decide the boolean and semantic decision, Analayze-Conflict does conflict analysis, then BackTrackAndDecide does backtracking and literal deciding.

7.3.1 Propagate

Propagate plays an essential role whole the NLSAT. The algorithm structure is similar to **UnitResolution+** [Algorithm 5] in SAT solver. Propagate does Boolean Resolution at the B-Resolution and runs a T-solver R-explain in the theory propagation process R-Propagate. Then it checks if there is conflict clause with B-Conflict and R-Conflict, while applying the propagation.

Algorithm 21 B-Propagate(M, F)

INPUT:A trail M, and a $\mathcal{T}$-formula where $\mathcal{T}$ is non linear real arithmetic
OUTPUT: A trail M' , if propagated literal found, added the literal, else M = M

for all $C = (l_i, \ldots, l_n, L) \in F$ **do**
  if $(L, \neg L) \notin M \wedge \forall l_i \in C\ (\neg l_i \in M)$ **then**
    $M' := M \cup L_{\downarrow C}$
    return M' ▷ If one propagated literal found, it returns
  end if
end for
return M

Algorithm 22 B-Conflict(M, F)

INPUT:A trail M, and a $\mathcal{T}$-formula where $\mathcal{T}$ is non linear real arithmetic
OUTPUT: A conflict clause A, if no conflict, it is $\emptyset$

$A := \emptyset$
for all $C \in F$ **do**
  if $\forall L \in C(\neg L \in M)$ **then**
    $A := C$
    return A
  end if
end for
return A ▷ no conflict found

To define R-Propagate, we first define R-feasible which checks the nonlinear real arithmetic level feasibility at the trail using **Base** [Algorithm 14].

Algorithm 23 R-feasible(M)

INPUT: a trail M
OUTPUT: $\mathtt{true}$ if all the mono variant polynomial constrains have any region to assign a value to the variable, else false

P := a set of mono variant polynomial constraints in M
for all $F \subset P$ such that each polynomial $f \in F$ has the same variable **do**
  $\mathcal{R}^1$:= Base(F)
  x := the variable of F
  if with the roots bounds for x in $\mathcal{R}^1$, $\exists f \in F$ is false **then**
    return false
  end if
end for
return true

R-feasible calls **Base** to get the roots and the intervals, then calculate the bound of x and check the semantic (nonlinear real arithmetic) literals in M whether there is some regions to assign x or not. If no region found for some variable x, R-feasible returns false.

Algorithm 24 R-Propagate(M, F)

INPUT:A trail M, and a $\mathcal{T}$-formula where $\mathcal{T}$ is non linear real arithmetic
OUTPUT:A trail M', if propagated literal found, added the literal, else $M' = M$

for all $L \in C \wedge$ if L or $\neg L$ is evaluated to undefined in M **do**
  if L is evaluated to be false in $M \cup L$ **then**
    $k :=$ is the highest level of assignment to evaluate $L = \mathtt{false}$
    $M' := M \cup L_{\downarrow k}$ ▷ Semantic propagation
    return M'
  else if $\neg$ R-feasible($M \cup L$) **then**
    $E :=$ R-Explain($M \cup L$) ▷ If $M \cup L$ is not feasible $M \cup L$ conflicts
    $M' := M \cup L_{\downarrow E}$
    return M'
  end if
end for
return M

Algorithm 25 R-Conflict(M, F)

INPUT:A trail M, and a $\mathcal{T}$-formula where $\mathcal{T}$ is non linear real arithmetic
OUTPUT: A conflict clause A, if no conflict, it is $\emptyset$

$A := \emptyset$
if $\neg$ R-feasible(M) **then**
  $E :=$ R-explain(M) ▷ Conflict find in M
  $A := E$
  return A
end if
return A ▷ No conflict found

Algorithm 26 Propagate(M, F)

INPUT: A trail M, and a $\mathcal{T}$-formula F where $\mathcal{T}$ is non linear real arithmetic.
OUTPUT: A trail M' if propagated literal found, added the literal, and
a conflict clause A, if no conflict, it is $\emptyset$.
while true **do**
  $M' :=$ B-Propagate(F, M)
  if $M == M'$, no boolean decision **then**
    $M' :=$ R-Propagate(F, M)
  end if
  $A :=$ B-Conflict(F, M)
  if $A \neq \emptyset$, their is a conflict **then**
    return M', A
  end if
  $A :=$ R-Conflict(F, M)
  if $A \neq \emptyset$, their is a real arithmetic conflict **then**
    return M', A
  end if
end while
return $M', A := \emptyset$

7.3.2 Decide

Since the trail in NLSAT has two types of decision: boolean decision and semantic decision, we have two types of deciding the value procedure: B-Decide, and R-Decide.

Algorithm 27 B-Decide(L, M)

INPUT:An boolean literal L, and a trail M
OUTPUT: An boolean literal L with the level

lv is the current highest level in trail M
$lv := lv + 1$ ▷ increase the level
$M' := M \cup \overset{lv}{L}$

return M'

B-Decide increase the level lv and add the literal L into the trail M with the level. The literal itself is already selected before we call B-Decide, because the way we decide the literal is different by the context.

Algorithm 28 R-Decide(x, M, F)

INPUT: An algebraic number variable x, a trail M, and a $\mathcal{T}$-formula F where $\mathcal{T}$ is non linear real arithmetic.
OUTPUT:An algebraic number variable x with the level.

lv is the current highest level in trail M
$l :=$ is the lower bound of x
$h :=$ is the higher bound of x
$V_d := \{v \mid x \neq v\}$
while true **do**
  $v :=$ find a value $(l < v < h) \wedge v \notin V_d$
end while

$lv := lv + 1$ ▷ increase the level
return $x \overset{lv}{\mapsto} v$, where the level lv is marked over the value

To get x value, NLSAT always maintains all the variables of mono polynomial literal in M. It tracks

- the lower bound of x by mono variant polynomial constraint $L \in M$,
- the uppper bound of x by mono variant polynomial constraint $L \in M$,
- the set V_d such that $V_d\{v \mid x \neq v\}$ by mono variant polynomial constraint $L \in$M.

If it is possible, NLSAT always set the x be a dyadic rationals.

7.3.3 AnalyzeConflict

The role of AnalyzeConflict is to create a new clause for the subsequent backtracking process.

Algorithm 29 AnalyzeConflict(M, C)

INPUT:INPUT: a trail M, and a clause C is conflicted in M
OUTPUT: Conflicting clause R either boolean conflict $R_{bConflict}$ or semantic conflict $R_{sConflict}$ that is usable for backtracking

$R := C$
$k :=$ is the length of M

while $R \neq \emptyset$ **do**
  if $M[k] = L_{\downarrow D} \wedge \neg L \in R$ **then**
    $\varphi := R \wedge (\neg D \vee L)$ ▷ ie $R = l_1 \vee \cdots \vee l_m \vee \neg L$, $(\neg D \vee L) = L_1 \vee \ldots L_n \vee L$
    $R :=$ get $l_1 \vee \cdots \vee l_m \vee L_1 \ldots L_n$ from φ ▷ Boolean resolution
  end if
  bCoflict := $\mathtt{true}$ if all the level of the literals in R is different
  sConflict := $\mathtt{true}$ if the highest level literals in R includes semantic propagation
  if bConflict **then**
    return R as $R_{bConflict}$ ▷ annotate it is boolean conflict
  else if sConflict **then**
    return R as $R_{sConflict}$ ▷ annotate it is semantic conflict
  end if
  $k := k - 1$
end while
return $R := \emptyset$ ▷ If comes here, nowhere to backtrack. Thus it is UNSAT

At the boolean resolution in the while loop, it creates a new clause that can be used for backtracking also for picking a new literal from the clause. Then it determines the conflict whether it is a boolean conflict or a semantic conflict. The if condition of bConflict and sConflict are required to do backtracking.

7.3.4 Backtracking and Clause Learning

Since the trail M contains not only boolean decision clause, the Backtracking and Clause Learning differ from basic $\mathsf{DPLL(T)}$ solvers. So we define BackTrackAndDecide.

Algorithm 30 BackTrackAndDecide(M, R)

INPUT: a trail M, and
a clause R is conflicted in M either boolean conflict $R_{\mathsf{bConflict}}$ or semantic conflict $R_{\mathsf{sConflict}}$ analyzed by AnalyzeConflict
OUTPUT: a backtracked trail M added new propagated literal or boolean decision

if R is a boolean conflict clause **then**
  $lv :=$ the second highest level in R
else if R is a semantic conflict clause **then**
  $lv := -1$ from the highest level of R
end if
$M' :=$ remove all elements in M the level $> lv$
if R is a boolean conflict clause **then**
  $L_{\downarrow R} := L \in R \wedge L \notin M$ ▷ By the conflict analysis it contains L that is not in M
  $M := M' \cup L_{\downarrow R}$
else if R is a semantic conflict clause **then**
  $L := L \in R \wedge L \notin M$
  $M :=$ B-Decide(L, M)
end if
return M

First, we use the level to do backtracking. The reason we introduce level is to do backtracking inside the trail.

Then we use the conflicting clause to get a literal. NLSAT does backtracking and decide at the same time [33, 34]. NLSAT may add a new literal that is not available in the original problem into the trail. It is because the given conflicting clause may contain a new literal by R-explain since AnalyzeConflict added some literals from the annotation clause of literals. The annotation clause is a set of original problems literals or the literals by the result of R-explain.

7.4 NLSAT algorithm

With the previous algorithms, the NLSAT algorithm structure is simple. The abstract structure is much similar to DPLL [Algorithm 6], than DPLL(T) [Algorithm 10].

NLSAT algorithm first try to do Propagate, if conflicting clause found, call AnalyzeConflict to do backtracking, and nowhere to backtrack, it is UNSAT, else calls BackTrackAndDecide. If Propagate does not find a conflicting clause, then Decide a boolean decision or semantic decision. If no more undecided boolean nor semantic value, it is SAT.

Algorithm 31 NLSAT(φ)

INPUT:A $\mathcal{T}$-formula φ where $\mathcal{T}$ is non linear real arithmetic
OUTPUT: SAT or UNSAT. If SAT, also returns satisfiable assignments

```
M := ∅                                                        ▷ a trail
F := φ
while true do
  M′, A := Propagate(M, F)
  if A ≠ ∅ then
    R := AnalyzeConflict(M, A) ▷ If Conflict clause A found, analyze it to backtrack

    if R = ∅ then
      return UNSAT, M =: ∅            ▷ If nowhere to backtrack, it is UNSAT
    end if
    M := BackTrackAndDecide(M, R)
  else
    if x := is a new variable ∈ F, x ∉ M then
      M := R-Decide(x, M, F)                     ▷ add the semantic decision
    else if L ∈ C ∈ F ∧ {L, ¬L} ∉ M then
      M := B-Decide(L, M)                         ▷ add the boolean decision
    else
      return SAT, M              ▷ eg ∀L ∈ C ∈ F({L ∨ ¬L} ∈ M) ∧ ∀x ∈ F(x ∈ M)
    end if
  end if
end while
```

Chapter 8

Conclusion

We studied the satisfiability algorithms for nonlinear real arithmetic problem, and gives the essential algorithms including polynomial. One of our research contribution is giving an overview of nonlinear real arithmetic satisfiability.

Furthermore, through investigating the NLSAT algorithm and the source code, we make sure the strategies for the performance among these topics:

- Variable selection
- Selection for the value of a semantic decision
- Eager or lazy in NLSAT
- The algorithm of Explain

We propose the way how we evaluate the alternative approaches among them.

Bibliography

[1] Bob F Caviness and Jeremy R Johnson. *Quantifier elimination and cylindrical algebraic decomposition*. Springer Science & Business Media, 2012.

[2] Alfred Tarski. A decision method for elementary algebra and geometry. *1948*, 1951.

[3] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 134–183. Springer, 1975.

[4] James H Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1):29–35, 1988.

[5] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[6] Joao Marques-Silva. Practical applications of boolean satisfiability. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 74–80. IEEE, 2008.

[7] Ashish Sabharwal. "modern sat solvers: Key advances and applications. *Technology Overview, IBM Watson Research Center*, 23, 2011.

[8] Van Khanh To and Mizuhito Ogawa. raSAT: SMT for Polynomial Inequality. *Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-2013-003: 1-23*, 2013.

[9] David Cox, John Little, and Donal O'shea. *Ideals, varieties, and algorithms*, volume 3. Springer, 1992.

[10] 硲文夫. 代数学. 森北出版, 1997.

[11] 野呂正行 and 横山和弘. グレブナー基底の計算基礎篇. 初版, 財団法人東京大学出版会, 2003.

[12] Serge Lang. Algebra revised third edition. *Graduate Texts in Mathematics*, 1(211): ALL–ALL, 2002.

[13] Richard Zippel. *Effective polynomial computation*, volume 241. Springer Science & Business Media, 2012.

[14] Emil Artin and Arthur Norton Milgram. *Galois theory*, volume 2. Courier Corporation, 1944.

[15] のんびり数学研究会. ガロアに出会う. 数学書房, 2014.

[16] Kazuhiro Yokoyama, Masayuki Noro, and Taku Takeshima. Computing primitive elements of extension fields. *Journal of Symbolic Computation*, 8(6):553–580, 1989.

[17] Barry M Trager. Algebraic factoring and rational function integration. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 219–226. ACM, 1976.

[18] Rüdiger Loos. Computing in algebraic extensions. In *Computer algebra*, pages 173–187. Springer, 1983.

[19] 高木貞治. 代数学講義改訂新版, 1965.

[20] Keith O Geddes, Stephen R Czapor, and George Labahn. *Algorithms for computer algebra*. Springer Science & Business Media, 1992.

[21] 穴井宏和 and 横山和弘. *QE の計算アルゴリズムとその応用: 数式処理による最適化*. 東京大学出版会, 2011.

[22] Saugata Basu, Richard Pollack, and Marie-Francoise Roy. *Algorithms in real algebraic geometry*, volume 20033. Springer, 2005.

[23] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract dpll and abstract dpll modulo theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 36–50. Springer, 2005.

[24] Randal E Bryant and Miroslav N Velev. Boolean satisfiability with transitivity constraints. *ACM Transactions on Computational Logic (TOCL)*, 3(4):604–627, 2002.

[25] Randal E Bryant, Shuvendu K Lahiri, and Sanjit A Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *International Conference on Computer Aided Verification*, pages 78–92. Springer, 2002.

[26] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[27] Michel Coste. *An introduction to semialgebraic geometry*. Citeseer, 2000.

[28] Mats Jirstrand. Cylindrical algebraic decomposition-an introduction. *Automatic Control group in Linköping*, 1995.

[29] Dennis S Arnon, George E Collins, and Scott McCallum. Cylindrical algebraic decomposition i: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.

[30] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[31] Dejan Jovanović and Leonardo De Moura. Solving non-linear arithmetic. *ACM Communications in Computer Algebra*, 46(3/4):104–105, 2013.

[32] Dejan Jovanovic. *SMT Beyond DPLL (T): A New Approach to Theory Solvers and Theory Combination*. PhD thesis, Courant Institute of Mathematical Sciences New York, 2012.

[33] Leonardo De Moura and Dejan Jovanović. A model-constructing satisfiability calculus. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 1–12. Springer, 2013.

[34] Dejan Jovanovic, Clark Barrett, and Leonardo De Moura. The design and implementation of the model constructing satisfiability calculus. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 173–180. IEEE, 2013.

[35] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in sat modulo theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 512–526. Springer, 2006.

[36] Bruno Dutertre and Leonardo De Moura. The yices smt solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, 2(2), 2006.

[37] Christopher W Brown. Constructing a single open cell in a cylindrical algebraic decomposition. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 133–140. ACM, 2013.

[38] Adnan Darwiche and Knot Pipatsrisawat. Complete algorithms. *Handbook of Satisfiability*, 185:99–130, 2009.

[39] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Conflict-driven clause learning sat solvers. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pages 131–153, 2009.

[40] Adam Strzeboński. Solving systems of strict polynomial inequalities. *Journal of Symbolic Computation*, 29(3):471–480, 2000.

[41] Michel Coste and Marie-Françoise Roy. Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets. *Journal of Symbolic Computation*, 5(1):121–129, 1988.

[42] Amir Pnueli, Yoav Rodeh, Ofer Shtrichman, and Michael Siegel. Deciding equality formulas by small domains instantiations. In *International Conference on Computer Aided Verification*, pages 455–469. Springer, 1999.