RECTIFICATION OF INTEGER ARITHMETIC CIRCUITS

by

Arpitha Srinath

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

The University of Utah

August 2019

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Arpitha Srinath

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Priyank Kalla

Pierre-Emmanuel Gaillardon

Chris Myers

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Arpitha Srinath in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Priyank Kalla
Chair, Supervisory Committee

Approved for the Major Department

FLORIAN SOLZBACHER
Chair/Dean

Approved for the Graduate Council

DAVID KIEDA
Dean of The Graduate School

ABSTRACT

ABSTRACT

Formal verification of arithmetic circuits checks whether or not a gate-level circuit correctly implements a given specification model. In recent years, verification of arithmetic circuits has received a lot of attention. However, rectification of arithmetic circuits has only been addressed very recently. This is an important problem that finds application not only in debugging, but also in engineering change orders and synthesis of approximate circuits. This master's thesis addresses the problem of *rectification of integer arithmetic circuits* using concepts from symbolic computer algebra.

When formal verification identifies the presence of a bug in the design, rectification needs to be performed to correct the function implemented by the circuit, so that it matches the given specification. Using concepts from symbolic computer algebra and algebraic geometry, we identify a net in the circuit such that, irrespective of the number or type of bugs/errors, the circuit can be rectified at that one particular net. This is called single-fix rectification. Once single-fix rectifiability of the circuit is ascertained, we compute a corresponding rectification function at the target net which corrects the circuit. The algorithmic approaches are based on the Gröbner basis algorithm, which is used as both a decision procedure (for ascertaining rectifiability) as well as a quantification procedure (for computing a rectification function). While this problem has been recently solved for finite field arithmetic circuits, this thesis investigates the application to integer arithmetic circuits, while presenting new theory, techniques and algorithmic implementations to rectify integer multiplier circuits.

For my parents, Sridevi and Srinath.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Hardware design typically starts with a high-level specification, given in terms of block diagrams, tables, and informal text that conveys the desired functionality. The specification undergoes a series of translations until a final design description in some high-level language [1] [2] is obtained. Validation of the design involves checking that the physical description conforms to the original specifications. In a traditional design flow, simulation is often the first choice to ensure such correctness. To achieve complete confidence in the design, this requires an exhaustive set of input patterns to excite all possible behaviors of the system and then analyzing the output values. Unfortunately, the increasing complexity of digital systems, coupled with a demand for short design cycles, has made it almost impossible to achieve complete simulation coverage. There is a non-linear relationship between design complexity and verification complexity [3], as shown in Fig. 1.1.

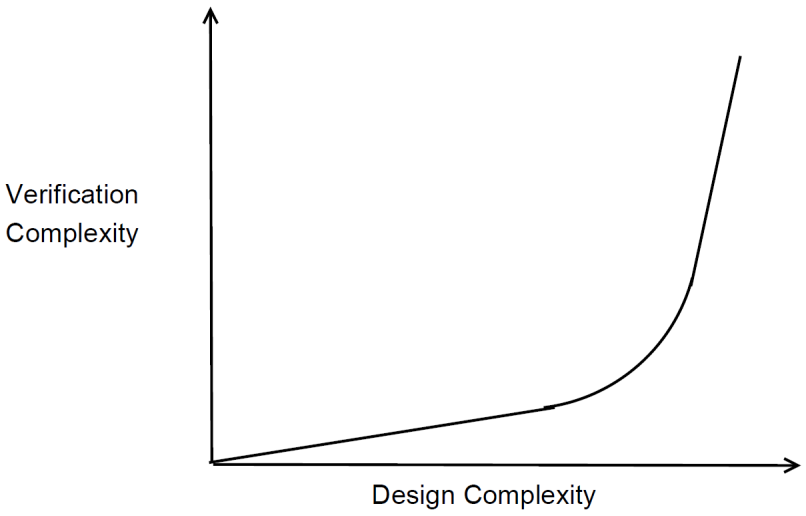

Figure 1.1: Design vs. Verification Complexity

Formal hardware verification attempts to overcome these weaknesses by trying to prove the correspondence between the implementation of the system and the desired specifications. The term *implementation* refers to the design description at any level of the hardware abstraction hierarchy. The term *specification* refers to the property with respect to which correctness is to be determined. In the long run, formal methods are more sophisticated in their approach to represent and analyze VLSI designs. The reason for this is because they exploit the powerful tools of mathematics rather than ad-hoc techniques.

1.1 Hardware Design Overview

The typical design flow of a hardware system, as shown in Fig. 1.2, begins with a hardware system specification, which describes the necessary functions and parameters that the system must perform and adhere to. The specification is typically modeled as a word-level polynomial. The polynomial is converted into a netlist, i.e., a large collection of small physical blocks (MOSFET, Boolean logic gates, etc.) and the interconnections (wires) between them. Lastly, the netlist is further optimized, then mapped onto a physical space on a chip, which can then be sent off for fabrication. Computer-Aided Design (CAD) tool automates this design flow.

1.1.1 Formal verification

In hardware design, moving from one abstraction level to another raises an important issue: how do we ensure that the functionality of the optimized design is equivalent to the original specification? If bugs in hardware design are not caught early, they can have costly effects like the need for a redesign. The two primary methodologies used in hardware verification are simulation and formal verification.

Formal verification is the process of utilizing mathematical theory to reason about the correctness of hardware designs. Formal verification in hardware usually takes one of two forms: property checking and equivalence checking. Property checking is a process of checking whether a design conforms to its given behavior

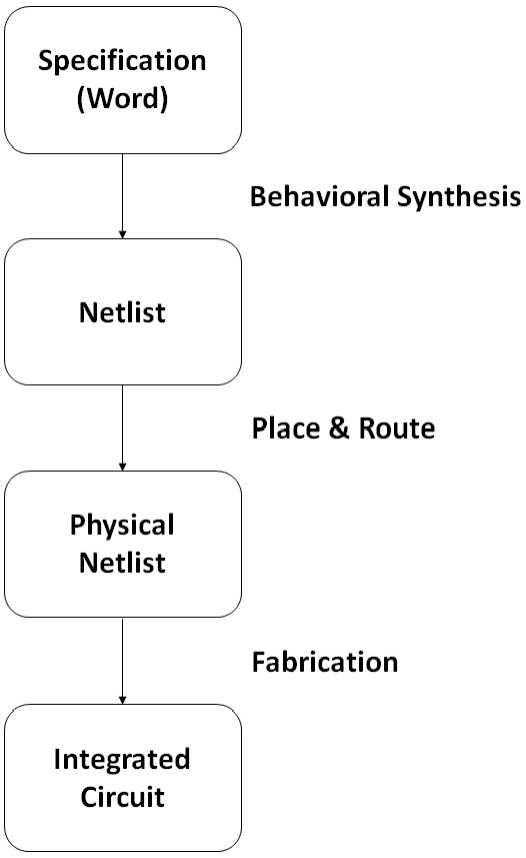

Figure 1.2: Typical hardware design flow

or properties. Equivalence checking is conducted to prove the equivalent functionality of two given designs, or to prove equivalence of design descriptions given at different levels of abstractions. Usually, equivalence checking is applied at various stages of the design cycle to verify the correctness of the applied transformations.

Techniques utilized by property checking include model checking, theorem proving, and approaches that integrate both. Equivalence checking makes use of Binary Decision Diagrams (BDDs), Satisfiability (SAT) solvers, and And-Inverter-Graph (AIG) based reductions, among others. In recent years, computer algebra based decision procedures are gaining popularity for equivalence checking of arithmetic datapaths. This kind of verification technique is believed to be more sophisticated in verifying arithmetic hardware designs in that they exploit the powerful applications of mathematics rather than ad-hoc techniques.

1.2 Formal verification of integer arithmetic circuits

Hardware designs contain a large number of custom-designed circuits such as adders, multipliers, dividers, and so on. Arithmetic units, i.e., adders and multipliers, are the most fundamental building blocks for all computing. Modern processors contain a wide variety of arithmetic circuits such as integer and floating-point units (adders, multipliers, dividers and so on) in addition to specialized systems (such as multiply-accumulate units). Such circuits are usually not synthesized by automatic synthesis tools, due to area and performance constraints, but rather are custom-designed. Therefore, this raises the potential for errors/bugs in the implementation.

Verification of arithmetic circuits has always been an essential part of processor verification. Traditional approaches to verifying arithmetic circuits based on simulation do not provide complete coverage. For example, the whole input space of each IEEE double-precision floating-point circuit with one rounding mode is 2^{128} test vectors, which is impossible to simulate in practice. Formal methods provide a natural framework for verifying such circuits, since they perform an abstraction of the underlying operations, thereby avoiding the generation of such a large number of test cases.

As an intensively investigated topic, techniques, and approaches for equivalence checking are well established. With various techniques employed for equivalence checking, BDDs [4] and SAT-based techniques [5] are the two dominant approaches widely used in both academia and industry. BDD-based approaches try to construct canonical representations of given circuits and conduct a linear comparison to determine whether they are equivalent or not. SAT-based equivalence checking approaches try to find the unsatisfiability of a miter representing two designs.

There are also many promising generalizations of SAT and BDDs: Binary Moment Diagrams (BMDs) [6] which have shown their superiority of verifying integer multipliers [6], and Satisfiability Modulo Theories (SMT) [7] solvers which are the next generation of SAT. These approaches, to some extent, have gained some successes in equivalence checking. However, these approaches are beginning to show

signs of inadequacy in two cases. First, large scale hardware designs still hinder the equivalence checking as the level of design complexity grows rapidly. For example, the verification of a 16-bit modular multiplier becomes infeasible for the current SAT/BDD based approaches, as shown by experiments in [8] [9]. Secondly, for structurally similar circuits, this problem can be efficiently solved using the techniques of AIG based reductions [10] and subsequent use of circuit-SAT solvers [11] [12]. However, when the circuits are functionally equivalent but structurally very dissimilar, none of the contemporary techniques, including BDDs, SAT and AIG-based approaches, are able to prove equivalence, as demonstrated by [8] [9].

Recent techniques have investigated the use of polynomial algebra and algebraic geometry techniques for their verification. The problem of verification of arithmetic circuits using symbolic algebra is well understood. These include verification of integer arithmetic circuits [13] [14] [15], integer modulo-arithmetic circuits [16], word-level RTL models of polynomial datapaths [17], [18] [19], finite field combinational circuits [20] [21] [22], and also sequential designs [23] [24]. The common approach used in contemporary literature is that the gates of the circuits are modeled as sets of polynomials in rings with coefficients from integers $\mathbb{Z}$, finite integer rings $\mathbb{Z}_{2^k}$, finite fields $\mathbb{F}_{2^k}$, and more recently also from the field of fractions $\mathbb{Q}$ [13]. Subsequently, the verification checks are formulated using algebraic geometry [25] (e.g., the Nullstellensatz), and *Gröbner basis (GB)* theory and technology [26] are used as decision procedures (ideal membership test) for formal verification.

1.3 On the rectification of integer multipliers

While the techniques discussed so far are successful in proving the correctness or detecting the presence of bugs, *the task of post-verification debugging, error diagnosis and rectification of arithmetic circuits has not been satisfactorily addressed*. Since arithmetic circuits are mostly custom-designed, it raises the potential for errors in the implementation, which (once detected) have to be eventually rectified. Instead of redesigning the entire circuit, it is desirable to synthesize rectification (sub-)functions with minimal topological changes to the existing design – a problem

often termed as *partial synthesis* [27] [28].

The debug, rectification and partial synthesis problem is analogous to that of *synthesis for Engineering Change Orders (ECO)*: the current circuit implementation (*Impl*) should be minimally modified (rectified) to match the ECO-updated specification (*Spec*). While ECO's may not be prevalent in arithmetic circuit design-flows, a similar setup can be used for *synthesis of approximate arithmetic circuits*: Given an efficient (say, area-optimal) VLSI implementation of an *exact* arithmetic circuit, rectify it with minimal topological changes so that it matches the approximate *Spec*.

The problem of debug, rectification and ECO synthesis has been addressed for control-dominated applications and random-logic circuits, where the early developments of [29] [30] [31] [32] were further extended by [33] [34] [27] by formulating debugging as a CNF-SAT problem, and computing rectification functions using *Craig Interpolants* [35] in propositional logic. Despite these efforts, contemporary approaches are infeasible for rectification of arithmetic circuits.

1.4 Thesis objective and scope

This thesis addresses the problem of rectification of buggy integer arithmetic circuits, specifically integer multipliers. Once formal verification detects the presence of bug(s) in the circuit, the next steps are (i) to perform debugging and diagnose the effects of those errors, and (ii) subsequently to compute a rectification function such that the rectified circuit implementation matches the specification.

This thesis focuses on rectification of hardware implementations of integer arithmetic circuits, specifically integer multipliers. Our objective is to perform functional rectification at a particular net in the circuit. Rectification is defined as the computation of a local sub-function for the buggy circuit such that, when this function is implemented in the circuit, the bug is rectified, and the circuit implementation and specification become functionally equivalent.

Rectification can be performed as either a single-fix or multi-fix. Single-fix rectification means that irrespective of the number/type of bugs, the function can be modified at a single net to correct the circuit. This thesis targets only single-fix rectification of integer multipliers. Multi-fix rectification is beyond the scope of

this thesis.

Once a bug is detected in the circuit, this thesis answers the question: Can the circuit be rectified at a single net location? Single-fix rectification for a bug may or may not exist. One of the objectives of this thesis is to identify a subset of internal nets where single-fix rectification may be possible. Subsequently, we identify a rectification target net. A rectification target is a net where its function can be modified such that the newly implemented function results in the circuit and specification becoming functionally equivalent. If single-fix rectification is not possible, our approach informs the users. On the other hand, if single-fix rectification is possible, we identify a net where rectification is indeed feasible. A rectification target net is identified, and our approach proves that our rectification function will indeed result in a correct implementation. The objective then is to compute a polynomial function (rectification polynomial) for this net, which can be synthesized as a sub-circuit to rectify the buggy implementation. This thesis also shows how to perform single-fix rectification and how to apply our approach in a CAD/synthesis framework. It shows experimental results conducted over a spectrum of integer multiplier circuits.

While the approach presented in this thesis is generic enough to rectify any arithmetic circuit against a polynomial specification, this thesis focuses on integer multipliers. Integer multipliers are the core component of computation systems, and their debug and rectification is more challenging. Therefore, experiments are conducted only on multiplier circuits.

1.5 Contributions of this Thesis

We propose computer-algebra and algebraic geometry techniques to perform post-verification debugging of arithmetic circuits, namely single-fix rectification. Our approach is formulated using Nullstellensatz principles from algebraic geometry, and the problem is algorithmically solved using Gröbner Bases. The gates of the given circuit implementation are modeled as a system of multivariate polynomials with coefficients from the field of rationals $\mathbb{Q}$. A Gröbner basis engine is utilized as both a decision procedure to ascertain rectifiability, and as a quantifi-

cation procedure to compute a polynomial function that can rectify the circuit at a single location.

The contributions of this thesis are enumerated as:

- Identify the primary outputs affected by the bug: This thesis describes how to analyze the result of verification to discern the outputs which are impacted by the bug.

- Identify a set of single-fix rectification targets: In theory, every net in the circuit can be checked for rectification. By identifying the outputs impacted by the bug, we describe how to simplify the search by identifying a subset of nets where single-fix may be feasible.

- Ascertain whether or not a net admits single-fix rectification: This thesis describes a way to confirm the existence of a rectification function that can be implemented at a given net to rectify the circuit. This rectification check *identifies the necessary and sufficient conditions* under which a net in the circuit admits single-fix rectification.

- Compute a rectification function: We describe a procedure to compute a rectification function that can be synthesized as a sub-circuit to rectify the implementation.

1.6 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 reviews previous applicable work and highlights their drawbacks with respect to rectification of circuits. Chapter 3 reviews preliminary concepts and notation related to computer-algebra, algebraic-geometry and Gröbner bases. Chapter 4 describes the model used to verify integer multiplier circuits against the given specification polynomial using Gröbner basis theory. Chapter 5 describes a way to obtain a subset of nets from the circuit where single-fix rectification is likely to be feasible. It subsequently describes how to select a net which admits single-fix rectification. Chapter 6 describes a procedure to compute a suitable rectification polynomial which, when

implemented, rectifies the circuit, so it exactly matches the given specification. It also shows the experimental results conducted over a range of integer multiplier circuits. Finally, Chapter 7 concludes the thesis with a perspective on current and future research directions on rectification of arithmetic circuits.

CHAPTER 2

PREVIOUS WORK ON RECTIFICATION

Rectification of buggy circuits is performed post verification/equivalence checking. The work of rectification lies at the crossroads of testing, verification, and logic synthesis. Therefore, rectification makes use of contemporary techniques from equivalence checking, as well as logic synthesis. Early techniques on rectification borrow concepts from Automatic Test Pattern Generation (ATPG) testing frameworks. Modern Engineering Change Order (ECO) synthesis framework culminates in QBF, Craig interpolation, and SAT-based approaches.

2.1 Contemporary approaches to rectification

Automated diagnosis and rectification of digital circuits have been addressed in [29, 30]. They used an ATPG framework to automate the diagnosis and the rectification of the design errors. Synthesis of Engineering Change Order (ECO) patches is an analogous problem to rectification. Engineering Change Orders are used to apply late-stage specification changes and bug fixes. The paper [31] presents algorithms for automatically synthesizing ECO patches. The use of Craig Interpolants for ECO has been presented in [33, 36, 37]. The single-fix rectification function approach in [36, 37] has been extended in [33] to multi-fix rectification. The problem of multi-fix rectification is solved by generating multiple partial-fix functions and incrementally and iteratively rectifying the circuit. The techniques guarantee that the number of error minterms/counter examples reduces at each step.

The unknown component problem is where a part of the circuit is unknown as a black box and needs to be synthesized. Synthesis of an unknown component in a large circuit is also called the partial synthesis problem. The approaches in [27], [34] resolves the unknown component problem using an incremental SAT

formulation. The paper [34] models the unknown component in a given circuit (Ckt) as a lookup table (LUT) by using transformation variables (X). The solution to these variables implements the desired logic function so that the resulting circuit becomes logically equivalent to a given specification $Spec()$. Let $Ckt(X, In)$ be the formula corresponding to the given circuit with possible transformations, where In is the set of all primary inputs to the circuit. This can be formulated naturally as a two-level QBF with an existential quantifier followed by a universal quantifier, as shown below:

$$\exists X.\forall In.\ Ckt(X,In) = Spec(In)\ : \tag{2.1}$$

The two-level QBF in Eqn. (2.1) is solved using a QBF solver in [32]. This work was extended in [38] by imposing restrictions on the immediate inputs of the unknown component. These constraints make the problem more complicated, and it is formulated as a Dependency Quantified Boolean Formulae (DQBF). However, as solving two-level QBF is infeasible for complicated circuits, [34] solve the QBF by repeated applications of Boolean SAT solvers.

The work in [39] poses the unknown component formulation as a camouflaged circuit model and tries to de-obfuscate several types of camouflaging techniques using incremental SAT solving. The approach used in [40] inserts logic corrector MUXs on the unknown sub-circuits and relies on SAT solvers to realize the functionality.

As these approaches are SAT-based, they work well for random logic circuits but are not efficient for arithmetic circuits. Despite using state of the art SAT solvers, all the above approaches fail to verify large and complex arithmetic circuits. The solvers still model the problem as decision procedures and, as demonstrated by experimental results, are shown to be inefficient in solving verification problems on multiplier circuits beyond 12-bits.

2.2 Rectification using computer algebra techniques

In recent years, computer algebra has been utilized for circuit debugging and rectification. Rectification of integer circuits is attempted in [41] [42]. However,

their approach is incomplete. Their approaches rely heavily on the arithmetic structure of the circuit for coefficient calculation. If the arithmetic circuits contain redundancy in the structure, their approach cannot identify the rectification targets. The rectification attempt in [43] also suffers from the same limitation.

The authors in [44] [45] have addressed the problem of single-fix rectification for finite field arithmetic circuits. They introduce the notion of Craig Interpolation over finite fields. Using the concept of Nullstellensatz, they identify target nets and compute the rectification function using Craig Interpolants. As an alternative to Craig Interpolation, [46] used Gröbner Bases concepts like ideal membership and extended Gröbner bases theory to compute a rectification function. However, these papers address the problem over finite fields, where Craig Interpolants are well defined due to the finiteness of the fields. But they do not address integer arithmetic circuits. This thesis draws inspiration from the verification model of [13] and also from the work of [46] for rectification.

CHAPTER 3

PRELIMINARIES AND NOTATION

This chapter gives an account of basic communicative algebra objects, such as groups, rings, fields and polynomials and their applications in algebraic geometry. The material of this chapter is mostly referred from the textbooks [25] [26].

3.1 Rings, Fields and Polynomials

Definition 3.1. *An **abelian group** is a set* $\mathbb{S}$ *and a binary operation 'add (+)' satisfying:*

- *Closure Law: For every* $a, b \in \mathbb{S}, a + b \in \mathbb{S}$.
- *Associative Law: For every* $a, b, c \in \mathbb{S}, a + (b + c) = (a + b) + c$.
- *Commutativity: For every* $a, b \in \mathbb{S}, a + b = b + a$.
- *Existence of Identity: There is an identity element* $0 \in \mathbb{S}$ *such that for all* $a \in \mathbb{S}, a + 0 = a$.
- *Existence of Inverse: If* $a \in \mathbb{S}$, *then there is an element* $a^{-1} \in \mathbb{S}$ *such that* $a + a^{-1} = 0$.

The set of integers $\mathbb{Z}$, for instance, forms an abelian group under addition.

Definition 3.2. *Given two binary operations 'add (+)' and 'multiply (·)' on the set* $\mathbb{R}$, *as well as two distinguished elements* $0, 1 \in \mathbb{R}$, *the system* $\mathbb{R}$ *is called a ring if the following properties hold:*

- $\mathbb{R}$ *forms an abelian group under the '+' operation with additive identity element 0.*
- *Distributive Laws: For all* $a, b, c \in \mathbb{R}$, $a \cdot (b + c) = a \cdot b + a \cdot c$.
- *Associative Law of Multiplication: For every* $a, b, c \in \mathbb{R}$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

If there is an identity element $1 \in \mathbb{R}$ *such that for all* $a \in \mathbb{R}$, $a \cdot 1 = a = 1 \cdot a$, *then* $\mathbb{R}$ *is said to be a* ***ring with unity****. The ring* $\mathbb{R}$ *is* ***commutative*** *if the following law also holds:*

- *Commutative Law of Multiplication: For every* $a, b \in \mathbb{R}$, $a \cdot b = b \cdot a$.

For the purpose of this thesis, any time we refer to a **ring**, we are specifically referring to a **commutative ring with unity**. Two common examples of such rings are the set of integers, $\mathbb{Z}$, and the set of rational numbers, $\mathbb{Q}$.

Definition 3.3. *A* ***field*** $\mathbb{F}$ *is a commutative ring with unity, where every non-zero element in* $\mathbb{F}$ *has a multiplicative inverse; i.e.* $\forall a \in \mathbb{F} - \{0\}, \exists\, \hat{a} \in \mathbb{F}$ *such that* $a \cdot \hat{a} = 1$.

A field is defined over a ring with an extra condition: the presence of a multiplicative inverse for all non-zero elements. It is a set on which addition, subtraction, multiplication, and division are defined. A field must be a ring while a ring is not necessarily a field. For example, the set $\mathbb{Z}_{2^k} = \{0, 1, \ldots, 2^k - 1\}$ forms a finite ring, where '+' and '$\cdot$' are operations add and multiply modulo 2^k. However, $\mathbb{Z}_{2^k}$ is not a field because not every element in $\mathbb{Z}_{2^k}$ has a multiplicative inverse. In the ring $\mathbb{Z}_{2^3}$, for instance, the element 5 has an inverse $(5 \cdot 5(mod\ 8) = 1)$ but the element 4 does not. In general, fields can be infinite, or contain a finite number of elements. For example, fractions $\mathbb{Q}$, complex numbers $\mathbb{C}$, are infinite fields. And $\mathbb{Z}_p$, where p is any prime number, are finite fields. In this thesis, we use the notation $\mathbb{F}$ to denote any arbitrary field. $\mathbb{Q}$ is the field of rational numbers, $\mathbb{R}$ is the field of real numbers and $\mathbb{Z}$ is the set of integers.

The main concept of field theory is **Field Extensions**. A field extension is a pair of fields $\mathbb{F} \subseteq \overline{\mathbb{F}}$ such that the operations of $\mathbb{F}$ are those of $\overline{\mathbb{F}}$ restricted to $\mathbb{F}$. In this case, $\overline{\mathbb{F}}$ is an extension field of $\mathbb{F}$ and $\mathbb{F}$ is a subfield of $\overline{\mathbb{F}}$. This larger field also satisfies additional properties.

Definition 3.4. *A field* $\overline{\mathbb{F}}$ *is an* **algebraically closed** *field if every polynomial in one variable with degree at least* 1, *with coefficients in* $\overline{\mathbb{F}}$, *has a root in* $\overline{\mathbb{F}}$.

In other words, any non-constant polynomial equation with coefficients from $\overline{\mathbb{F}}$ will always has at least one root in $\overline{\mathbb{F}}$. Every field $\mathbb{F}$ is contained in an algebraically

closed one $\overline{\mathbb{F}}$. For example, the field of real numbers $\mathbb{R}$ is not an algebraically closed field, because $x^2 + 1 = 0$ has no root in $\mathbb{R}$. However, $x^2 + 1 = 0$ has roots in the field of complex numbers $\mathbb{C}$, which is an algebraically closed field. In fact, $\mathbb{C}$ is the algebraic closure of $\mathbb{R}$. Also, every algebraically closed field is an infinite field. In this thesis, we work over the field of the rational numbers $\mathbb{Q}$. Finite extensions of $\mathbb{Q}$ are called algebraic number fields, and the algebraic closure of $\mathbb{Q}$ is the field of algebraic numbers and is denoted by $\overline{\mathbb{Q}}$ in this thesis.

Definition 3.5. *A **monomial** in variables $x_1, \ldots, x_n$ is a power product of the form*

$$x_1^{e_1} \cdot x_2^{e_2} \ldots x_n^{e_n}$$

where $e_i \in \mathbb{Z}_{\geqslant 0}, i \in \{1, \ldots, n\}$. The total degree of the monomial is $e_1 + \cdots + e_n$.

Thus, $x^2 \cdot y$ is a monomial in variables x, y with total degree 3.

Definition 3.6. *A **polynomial** f in $x_1, \ldots, x_n$ with coefficients from any given field $\mathbb{F}$ is a finite linear combinations of monomials with coefficients in $\mathbb{F}$. A polynomial is written in the form*

$$f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t, \forall c_i \in \mathbb{F}.$$

The set of all polynomials in $x_1, x_2, \ldots, x_n$ with coefficients in field $\mathbb{F}$ is denoted by ring $R = \mathbb{F}[x_1, x_2, \ldots, x_n]$. Thus, $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$

- *X_i are the monomials of the polynomial f*
- *$c_i \in \mathbb{F}$ is coefficient of monomial $c_i X_i$*
- *If $c_i \neq 0$, $c_i X_i$ is a term of the polynomial f*

As an example, $2x^2 + y$ is a polynomial with two terms $2x^2$ and y, with 2 and 1 as coefficients respectively and x^2 and y as monomials.

To process polynomials systematically, a monomial order order $>$ (a term order) is imposed on the ring. This is a total order and a well-order on all the monomials of R.

Definition 3.7. *Let $T^d = \{x^\alpha : \alpha \in \mathbb{Z}^d{}_{\geqslant 0}\}$ be the set of all monomials in $x_1, \ldots, x_d$. A* ***monomial order*** *$>$ on T^d is a total well-ordering satisfying:*

- *One and only one of the following should be true:*
  $x^\alpha > x^\beta$ *or* $x^\alpha = x^\beta$ *or* $x^\alpha < x^\beta$
- $1 < x^\alpha, \forall x^\alpha \ (x^\alpha \neq 1)$
- $x^\alpha < x^\beta \implies x^\alpha \cdot x^\gamma < x^\beta \cdot x^\gamma$

Term-orderings are totally ordered and well-ordered (there is a least element), i.e. anti-symmetric with constant terms last in the ordering. A total-order ensures that there is no ambiguity with respect to where a term is found in the term-ordering. Total orderings for monomials come in different forms, notably lexicographic orderings (lex), and its variants: degree-lexicographic ordering (deglex) and reverse degree-lexicographic ordering (revdeglex).

A **lexicographic ordering** (lex) is a total-ordering $>$ such that variables in the terms are lexicographically ordered, i.e. simply based on when the variables appear in the ordering. Higher variable-degrees take precedence over lower degrees for equivalent variables (e.g. $a^3 > a^2$).

Definition 3.8. Lexicographic order: *Let $x_1 > x_2 > \cdots > x_d$ lexicographically. Also let $\alpha = (\alpha_1, \ldots, \alpha_d)$; $\beta = (\beta_1, \ldots, \beta_d) \in \mathbb{Z}^d_{\geq 0}$. Then we have:*

$$x^\alpha > x^\beta \iff \begin{cases} \text{Starting from the left, the first co-ordinates of } \alpha_i, \beta_i \\ \text{that are different satisfy } \alpha_i > \beta_i \end{cases} \tag{3.1}$$

In this work, we consider mostly lexicographic (lex) term orders.

Once a monomial order is imposed on R, the monomials of all polynomials $f = c_1 X_1 + c_2 X_2 + \cdots + c_t X_t$ are ordered with respect to $>$.

Definition 3.9. *The* **leading term** *is the first term in a term-ordered polynomial. Likewise, the* **leading coefficient** *is the coefficient of the leading term. Finally, a* **leading monomial** *is the leading term lacking the coefficient. We use the following notation:*

$$lt(f) \quad \text{— Leading Term} \tag{3.2}$$

$$lc(f) \quad \text{— Leading Coefficient} \tag{3.3}$$

$$lm(f) \quad \text{— Leading Monomial} \tag{3.4}$$

Example 3.1.

$$\text{Let } f = 3a^2b + 2ab + 4bc. \text{ Then,} \tag{3.5}$$

$$lt(f) = 3a^2b, \tag{3.6}$$

$$lc(f) = 3, \tag{3.7}$$

$$lm(f) = a^2b. \tag{3.8}$$

3.2 Ideals and Varieties

In computer-algebra based formal verification, it is often necessary to analyze the presence or absence of solutions to a given system of constraints. In our applications, these constraints are polynomial equations and their solutions are modelled as **varieties**.

Definition 3.10. *Let $\mathbb{F}$ be a field, and let $f_1, \ldots, f_s \in \mathbb{F}[x_1, x_2, \ldots, x_n]$. Then we call $V(f_1, \ldots, f_s)$ the* **affine variety** *defined by $f_1, \ldots, f_s$ as:*

$$V(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \subseteq \mathbb{F}^n : f_i(a_1, \ldots, a_n) = 0, \forall i \; s.t. \; 1 \leq i \leq s\} \tag{3.9}$$

$V(f_1, \ldots, f_s) \subseteq \mathbb{F}^n$ is **the set of all solutions** in $\mathbb{F}^n$ of the system of equations: $f_1(x_1, \ldots, x_n) = \cdots = f_s(x_1, \ldots, x_n) = 0$.

Example 3.2. *Given $\mathbb{R}[x, y]$, $V(x^2 + y^2)$ is the set of all elements that satisfy $x^2 + y^2 = 0$ over $\mathbb{R}^2$. So $V(x^2 + y^2) = \{(0, 0)\}$. Similarly, in $\mathbb{R}[x, y]$, $V(x^2 + y^2 - 1) = \{$all points on the circle with unit radius : $x^2 + y^2 - 1 = 0\}$. Note that varieties depend on which field we are operating on. For the same polynomial $x^2 + 1$, we have:*

- *In $\mathbb{R}[x]$, $V_{\mathbb{R}}(x^2 + 1) = \emptyset$.*
- *In $\mathbb{C}[x]$, $V_{\mathbb{C}}(x^2 + 1) = \{(\pm i)\}$.*

The above example shows the variety can be infinite, finite (non-empty set) or empty. One important distinction we need to make about varieties is that a variety

depends not just on the given system of polynomial equations, but rather on the **ideal** generated by the polynomials.

Definition 3.11. *A subset $J \subset R = \mathbb{F}[x_1, \ldots, x_n]$ is an ideal if it satisfies:*

- $0 \in J$.
- *If $f, g \in J$, then $f + g \in J$.*
- *If $f \in J$ and $h \in R$, then $hf \in J$.*

Example 3.3. *The set of even integers, which is a subset of the ring of integers $\mathbb{Z}$, forms an ideal of $\mathbb{Z}$. This can be seen from the following;*

- 0 *belongs to the set of even integers.*
- *The sum of two even integers x and y is always an even integer.*
- *The product of any integer x with an even integer y is always an even integer.*

Sets of polynomials also generate ideals. Such ideals are referred to as polynomial ideals.

Definition 3.12. *Let $F = \{f_1, \ldots, f_s\}$ be polynomials in $R = \mathbb{F}[x_1, \ldots, x_n]$. Then we set*

$$\begin{aligned} J = \langle f_1, \ldots, f_s \rangle &= \{h_i f_1 + \cdots + h_s f_s : h_1, \ldots, h_s \in R\} \\ J = \langle f_1, \ldots, f_s \rangle &= \{\sum_{i=1}^{s} h_i f_i : h_i \in R\} \end{aligned} \tag{3.10}$$

*Then $\langle f_1, \ldots, f_s \rangle$ is the **ideal** J generated by $\{f_1, \ldots, f_s\}$. The polynomials F form the basis or the generators of J.*

Example 3.4. *Given $\mathbb{R}[x, y]$, $I = \langle x, y \rangle$ is an ideal containing all polynomials generated by x and y, such as $x^2 + y$ and $x + x \cdot y$. $J = \langle x^2, y^2 \rangle$ is an ideal containing all polynomials generated by x^2 and y^2, such as $x^2 + y^3$ and $x^{10} + x^2 \cdot y^2$. Notice that $J \subset I$ because every polynomial generated by J can be generated by I. But $I \neq J$ because $x + y$ can only be generated by I.*

When reasoning about variety of ideals, varieties are usually considered over the algebraic closure of the field we are working over. While working over the field of fractions $\mathbb{Q}$, we define varieties over its algebraic closure $\overline{\mathbb{Q}}$. In this thesis, we simplify notation by writing $V(f_1, \ldots, f_s)$ to mean $V_{\overline{\mathbb{Q}}}(f_1, \ldots, f_s)$, the variety of the system of polynomials over the algebraic closure of the field we are working on. However, we will use the notation $V_{\mathbb{Q}}$ and $V_{\overline{\mathbb{Q}}}$ when we need to specify which field the variety is considered over.

The same ideal may have many different bases. For instance, it is possible to have different sets of polynomials $\{f_1, \ldots, f_s\}$ and $\{g_1, \ldots, g_t\}$ that may generate the same ideal, i.e., $\langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$.

Proposition 3.1. *If $f_1, \ldots, f_s$ and $g_1, \ldots, g_t$ are bases of the same ideal in $\mathbb{F}[x_1, \ldots, x_n]$, so that $\langle f_1, \ldots, f_s \rangle = \langle g_1, \ldots, g_t \rangle$, then $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$.*

Example 3.5. *Consider the two bases $F_1 = \{2x^2 + 3y^2 - 11, x^2 - y^2 - 3\}$ and $F_2 = \{x^2 - 4, y^2 - 1\}$. These two bases generate the same ideal, i.e., $\langle F_1 \rangle = \langle F_2 \rangle$. Therefore, they represent the same variety, i.e.,*

$$V(F_1) = V(F_2) = \{\pm 2, \pm 1\}. \tag{3.11}$$

Ideals and their varieties are a key part of computer-algebra based formal verification. A given hardware design can be transformed into a set of polynomials over a field, $f_1, \ldots, f_s \in \mathbb{F}$. This set of polynomials gives the system of equations:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_s(x_1, \ldots, x_n) &= 0 \end{aligned}$$

Using algebra, it is possible to derive new equations from the original system. The ideal $\langle f_1, \ldots, f_s \rangle$ provides a way of analyzing such *consequences* of a system of polynomials.

Example 3.6. *Given two equations in $\mathbb{R}[x, y, z]$:*

$$\begin{aligned} x &= z + 1 \\ y &= x^2 + 1 \end{aligned}$$

we can eliminate x to obtain a new equation:

$$y = (z+1)^2 + 1 = z^2 + 2z + 2$$

Let $f_1, f_2, h \in \mathbb{R}[x, y, z]$ be polynomials based on these equations:

$$\begin{aligned} f_1 = x - z - 1 &= 0 \\ f_2 = y - x^2 - 1 &= 0 \\ h = y - z^2 - 2z - 2 &= 0 \end{aligned}$$

If I is the ideal generated by f_1 and f_2, i.e. $I = \langle f_1, f_2 \rangle$, then we find $h \in I$ as follows:

$$h = h_1 \cdot f_1 + h_2 \cdot f_2 = y - z^2 - 2z - 2$$

where

$$\begin{aligned} h_1 &= x + z + 1 \\ h_2 &= 1 \\ \text{and } h_1, h_2 &\in \mathbb{R}[x, y, z] \end{aligned}$$

As h can be written as a combination of f_1 and f_2, we call h a **member of the ideal** *$I = \langle f_1, f_2 \rangle$.*

3.2.1 Ideals of Varieties

Let $\mathbb{F}$ be any field and let $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$ be a point, and $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. We say that *f vanishes* on $\mathbf{a}$ if $f(\mathbf{a}) = 0$, i.e., $\mathbf{a}$ is in the variety of f.

Definition 3.13. *For any variety V of $\mathbb{F}^n$, the ideal of polynomials that vanish on V, called the vanishing ideal of V, is defined as $I(V) = \{f \in \mathbb{F}[x_1, \ldots, x_n] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$.*

Proposition 3.2. *If a polynomial f vanishes on a variety V, then $f \in I(V)$.*

Example 3.7. *Let ideal $J = \langle x^2, y^2 \rangle$. Then $V(J) = \{(0,0)\}$. All polynomials in J will obviously agree with the solution and vanish on this variety. However, the polynomials x, y are not in J but they also vanish on this variety. Therefore, $I(V(J))$ is the set of all polynomials that vanish on $V(J)$, and the polynomials $x, y \in I(V(J))$.*

Definition 3.14. *Let $J \subset \mathbb{F}[x_1, \ldots, x_n]$ be an ideal. The radical of J is defined as $\sqrt{J} = \{f \in \mathbb{F}[x_1, \ldots, x_n] : \exists m \in \mathbb{N}, f^m \in J\}$.*

Example 3.8. *Let $J = \langle x^2, y^2 \rangle \subset \mathbb{F}[x, y]$. Note that neither x nor y belongs to J, but they belong to $\sqrt{J}$. Similarly, $(x \cdot y) \notin J$, but since $(x \cdot y)^2 = x^2 \cdot y^2 \in J$, therefore, $x \cdot y \in \sqrt{J}$.*

When $J = \sqrt{J}$, then J is said to be a *radical ideal*. Moreover, $I(V)$ is a radical ideal. By analyzing the ideal J, generated by a system of polynomials derived from a hardware design, its variety $V(J)$, and the ideal of vanishing polynomials over this variety, $I(V(J))$, we can reason about the existence of certain properties of the design. To check for the existence of a property, we formulate the property as a polynomial and then perform an **ideal membership test** to determine if this polynomial is contained within the ideal $I(V(J))$.

Definition 3.15. *(Sum and Product of ideals) If $I = \langle f_1, \ldots, f_r \rangle$ and $J = \langle g_1, \ldots, g_s \rangle$ are ideals in R, then the sum of I and J is defined as $I + J = \langle f_1, \ldots, f_r, g_1, \ldots, g_s \rangle$. Similarly, the product of I and J is $I \cdot J = \langle f_i g_j \mid 1 \leq i \leq r,\ 1 \leq j \leq s$.*

Definition 3.16. *(Union and Intersection of Varieties) If I and J are ideals in R, then $V(I + J) = V(I) \cap V(J)$ and $V(I \cdot J) = V(I) \cup V(J)$.*

Definition 3.17. *(Ideal of vanishing polynomials) Let $F_0 = \{x_l^2 - x_l : l = 1, \ldots, n\}$ be a set of polynomials over the ring $\mathbb{Q}[x_1, \ldots, x_n]$. Let J_0 be the ideal generated by F_0, $J_0 = \langle F_0 \rangle$. The variety of ideal J_0 is $V(J_0) = \{0, 1\}^n$.*

Consider the polynomial $f = x_l^2 - x_l \in Q$. The solution to this polynomial f over $\overline{\mathbb{Q}}$, and over $\mathbb{Q}$, is $\{0, 1\}$. Therefore, $V(J_0) = V_{\mathbb{Q}}(J_0) = V_{\overline{\mathbb{Q}}}(J_0) = \{0, 1\}^n$.

The field of fractions $\mathbb{Q}$ is an infinite field. But $V(J_0) \subseteq \mathbb{Q}^n$ is a finite set of points. Let $J = \langle f_1, \ldots, f_s \rangle \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ be an arbitrary ideal and $J_0 \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Following Def. 3.16, $V(J + J_0) = V(J) \cap V(J_0)$. This implies that $V(J + J_0)$ is a finite set of points. Ideals like $J + J_0$ with a finite number of solutions are called **zero-dimensional ideals.**

We use the results of Hilbert's Nullstellensatz to establish correspondence between ideals and varieties in the context of algebraic geometry.

Theorem 3.1. *(Strong Nullstellensatz [26]) Let $\overline{\mathbb{F}}$ be an algebraically closed field, and let J be an ideal in $\overline{\mathbb{F}}[x_1, \ldots, x_n]$. Then we have $I(V_{\overline{\mathbb{F}}}(J)) = \sqrt{J}$.*

3.3 Gröbner Bases

As mentioned earlier, different polynomial sets may generate the same ideal. Some of these generating sets may be a better representation of the ideal, and thus provide more information and insight into the properties of ideal. One such ideal representation is called a **Gröbner basis**, which has a number of important properties that can solve numerous polynomial decision questions. In general, Gröbner Basis can help us ascertain

- Presence or absence of solutions (varieties)
- Dimension of the varieties
- Membership of a polynomial in an ideal

In essence, a Gröbner basis is a canonical representation of an ideal. There are many equivalent definitions of Gröbner bases, so we start with the definition that best describes their properties:

Definition 3.18. *A set of non-zero polynomials $G = \{g_1, \ldots, g_t\}$ which generate ideal $J = \langle g_1, \ldots, g_t \rangle$, is called a* **Gröbner basis** *for J if and only if for all $f \in J$, where $f \neq 0$, there exists a $g_i \in G$ such that $lm(g_i)$ divides $lm(f)$.*

$$G = GröbnerBasis(J) \iff \forall f \in J : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f) \tag{3.12}$$

The foundation for computing the Gröbner basis of an ideal was laid out by Buchberger [47]. Given a set of polynomials $F = \{f_1, \ldots, f_s\}$ that generate ideal $J = \langle f_1, \ldots, f_s \rangle$, Buchberger gives an algorithm to compute a Gröbner basis $G = \langle g_1, \ldots, g_t \rangle$. This algorithm relies on the notions of S-polynomials and polynomial reduction.

Definition 3.19. *For $f, g \in \mathbb{F}[x_1, \ldots, x_d]$, an* **S-polynomial** *$Spoly(f, g)$ is defined as:*

$$Spoly(f,g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g \tag{3.13}$$
$$\text{where } L = LCM\,(lt(f), lt(g))$$

Note LCM denotes least common multiple.

Definition 3.20. *Let $f, f_1, \ldots, f_s, r \in F[x_1, \ldots, x_n], f_i \neq 0; F = \{f_1, \ldots, f_s\}$. Then f reduces to r modulo F:*

$$f \xrightarrow{F}_+ r$$

if and only if there exists a sequence of indices $i_1, i_2, \ldots, i_t \in \{1, \ldots, s\}$ and a sequence of polynomials $r_1, \ldots, r_{t1} \in \mathbb{F}[x_1, \ldots, x_n]$ such that

$$f \xrightarrow{f_{i_1}} r_1 \xrightarrow{f_{i_2}} r_2 \xrightarrow{f_{i_3}} \ldots \xrightarrow{f_{t-1}} r_{t-1} \xrightarrow{f_{i_t}} r$$

Definition 3.21. *If $f \xrightarrow{F}_+ r$, then no term in r is divisible by $lt(f_i), \forall f_i \in F$. Then r is reduced w.r.t. F and it is called the remainder.*

The reduction process $f \xrightarrow{F}_+ r$, of dividing a polynomial f by a set of polynomials of F, can be modeled as repeated long-division of f by each of the polynomials in F until no further reductions can be made. This reduction process is shown in Algorithm 1.

Algorithm 1: Polynomial Division

Input: $f, f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n], f_i \neq 0$
Output: $u_1, \ldots, u_s, r$, s.t $f = \sigma f_i u_i + r$ where r is reduced w.r.t.
$F = \{f_1, \ldots, f_s\}$
$u_i \leftarrow 0$; $r \leftarrow 0$, $h \leftarrow f$;
while $h \neq 0$ **do**
  if $\exists i$ *s.t* $lm(f_i) \mid lm(h)$ **then**
    choose i least s.t. $lm(f_i) \mid lm(h)$
    $u_i = u_i + \frac{lt(h)}{lt(f_i)}$
    $h = h - lt(h) - \frac{lt(h)}{lt(f_i)} f_i$
  else
    $r = r + lt(h)$
    $h = h - lt(h)$
  end
end

The division algorithm keeps canceling the leading terms of polynomials until no more leading terms can be further canceled. So the key step is $p = p - lt(p)/lt(f_i) \cdot f_i$, as the following example shows.

Example 3.9. *Given $f = y^2 - x$ and $f_1 = y - x$ in $\mathbb{Q}[x, y]$ with deglex: $y > x$, perform $f \xrightarrow{f_1}_+ r$:*

1. $f = y^2 - x$, $f/f_1 = f - lt(f)/lt(f_1) \cdot f_1 = y^2 - x - (y^2/y) \cdot (y - x) = y \cdot x - x$
2. $f = y \cdot x - x$, $f/f_1 = f - lt(f)/lt(f_1) \cdot f_1 = (y \cdot x - x)/f_1 = x^2 - x$
3. $f = x^2 - x$, *no more operations possible, so* $r = x^2 - x$

With the notions of S-polynomials and polynomial reduction in place, we can now present Buchberger's Algorithm for computing Gröbner bases [47]. Note that a fixed monomial (term) ordering is required for a Gröbner basis computation to ensure that polynomials are manipulated in a consistent manner.

Algorithm 2: Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$, such that $J = \langle f_1, \ldots, f_s \rangle$
Output: $G = \{g_1, \ldots, g_t\}$, a Gröbner basis of I
$G := F$;
repeat
  $G' := G$;
  for *each pair* $\{f_i, f_j\}, i \neq j$ *in* G' **do**
    $Spoly(f_i, f_j) \xrightarrow{G'}_+ r$;
    if $r \neq 0$ **then**
      $G := G \cup \{r\}$;
    end
  end
until $G = G'$;

Buchberger's algorithm takes pairs of polynomials (f_i, f_j) in the basis G and combines them into "S-polynomials" ($Spoly(f_i, f_j)$) to cancel leading terms. The S-polynomial is then reduced (divided) by all elements of G to a remainder r, denoted as $Spoly(f_i, f_j) \xrightarrow{G}_+ r$. This process is repeated for all unique pairs of polynomials, including those created by newly added elements, until no new polynomials are generated; ultimately constructing the Gröbner basis.

Example 3.10. *Consider ideal $I \subset \mathbb{Q}[x,y]$, $I = \langle f_1, f_2 \rangle$, where $f_1 = yx - y$, $f_2 = y^2 - x$. Assume a degree-lexicographic term ordering with $y > x$ is imposed.*

First, we need to compute $Spoly(f_1, f_2) = x \cdot f_2 - y \cdot f_1 = y^2 - x^2$. Then we conduct a polynomial reduction $y^2 - x^2 \xrightarrow{f_2} x^2 - x \xrightarrow{f_1} x^2 - x$. Let $f_3 = x^2 - x$. Then G is updated as $\{f_1, f_2, f_3\}$. Next we compute $Spoly(f_1, f_3) = 0$. So there is no new polynomial generated. Similarly, we compute $Spoly(f_2, f_3) = x \cdot y^2 - x^3$, followed by $x \cdot y^2 - x^3 \xrightarrow{f_1} y^2 - x^3 \xrightarrow{f_2} x - x^3 \xrightarrow{f_2} 0$. Again, no polynomial is generated. Finally, $G = \{f_1, f_2, f_3\}$.

When computing a Gröbner basis, it's important to note that if $lt(f_i)$ and $lt(f_j)$ have no common variables, the Spoly reduction step in Buchberger's algorithm, $Spoly(f_i, f_j) \xrightarrow{G'}_+ r$, will produce $r = 0$.

Proof. If $lt(f)$ and $lt(g)$ have no common variables, $L = lcm(lt(f), lt(g)) = lt(f) \cdot lt(g)$. Then:

$$\begin{aligned} Spoly(f,g) &= \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g = \frac{lt(f) \cdot lt(g)}{lt(f)} \cdot f - \frac{lt(f) \cdot lt(g)}{lt(g)} \cdot g \\ &= lt(g) \cdot f - lt(f) \cdot g \end{aligned}$$

Thus, every monomial in $Spoly(f,g)$ is divisible by either $lt(f)$ or $lt(g)$, so computing $Spoly(f,g) \xrightarrow{f,g}_+ r$ will give $r = 0$. ■

As mentioned previously, a Gröbner basis gives a decision procedure to test for polynomial membership in an ideal. This is explained in the following Theorem.

Theorem 3.2. *(Ideal Membership Testing) Let $G = GB(J) = \{g_1, \ldots, g_t\}$ be the Gröbner basis of ideal J, and f be any polynomial. Then $f \in J \iff f \xrightarrow{G=\{g_1,\ldots,g_t\}}_+ 0$.*

Example 3.11. *Consider Example 3.10. Let $f = y^2x - x$ be another polynomial. Note that $f = yf_1 + f_2$, so $f \in I$. If we divide f by f_1 first and then by f_2, we will obtain a zero remainder. However, since the set $\{f_1, f_2\}$ is not a Gröbner basis, we find that the reduction $f \xrightarrow{f_2} x^2 - x \xrightarrow{f_1} x^2 - x \neq 0$; i.e. dividing f by f_2 first and then by f_1 does not lead to a zero remainder. However, if we compute the Gröbner basis G of I, $G = \{x^2 - x, yx - y, y^2 - x\}$, dividing f by polynomials in G in any order will always lead to the zero remainder. Therefore, one can decide ideal membership unequivocally using the Gröbner basis.*

If $f \in J$, f can be written as a linear combination (with polynomial coefficients) of the elements of the Gröbner basis:

$$f = u_1 g_1 + u_2 g_2 + \cdots + u_t g_t, \tag{3.14}$$

where u_i's correspond to the quotients of division $f \xrightarrow{g_1,\ldots,g_t}_+ 0$.

Buchberger's algorithm can be easily extended to output not just the Gröbner basis $G = \{g_1, \ldots, g_t\}$ but also a $t \times s$ matrix M with polynomial entries such that:

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_t \end{bmatrix} = M \cdot \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \end{bmatrix} \tag{3.15}$$

Subsequently, Eqns. (3.14) and (3.15) can be combined to give f as combination of the original polynomials $f_1, \ldots, f_s$:

$$f = v_1 f_1 + \cdots + v_s f_s. \tag{3.16}$$

A Gröbner basis may contain redundant elements. These redundancies can be removed by computing a **reduced Gröbner basis**. To compute a reduced Gröbner basis, we first must compute a minimal Gröbner basis.

Definition 3.22. *A* **minimal Gröbner basis** *for a polynomial ideal J is a Gröbner basis G for J such that*

- $lc(g_i) = 1, \forall g_i \in G$
- $\forall g_i \in G, lt(g_i) \notin \langle lt(G - \{g_i\}) \rangle$

A **minimal** Gröbner basis is a Gröbner basis such that all polynomials have a coefficient of 1 and no leading term of any element in G divides another in G. Given a Gröbner basis G, a minimal Gröbner basis can be computed as follows:

1. Minimize every $g_i \in G$, i.e $g_i = g_i / lc(g_i)$
2. For $g_i, g_j \in G$ where $i \neq j$, remove g_i from G if $lt(g_i) / lt(g_j)$, i.e. remove every polynomial in G whose leading term is divisible by the leading term of some other polynomial in G.

A minimal Gröbner basis can then be further reduced.

Definition 3.23. *A reduced Gröbner basis for a polynomial ideal J is a Gröbner basis $G = \{g_1, \ldots, g_t\}$ such that:*

- $lc(g_i) = 1, \forall g_i \in G$
- $\forall g_i \in G$, *no monomial of* g_i *lies in* $\langle lt(G - \{g_i\})\rangle$

G is a reduced Gröbner basis when no monomial of any element in G divides the leading term of another element.

For a given monomial ordering, the reduced Gröbner basis is a canonical representation of the ideal, as given by Proposition 3.3 below from [26].

Proposition 3.3. *Let $J \neq \{0\}$ be a polynomial ideal. Then, for a given monomial ordering, J has a unique reduced Gröbner basis.*

3.4 Complexity of Gröbner Basis

Let ideal $J = \langle f_1, \ldots, f_s\rangle \subseteq \mathbb{F}[x_1, \ldots, x_n]$ be any ideal. $\mathbb{F}[x_1, \ldots, x_n]$ is a ring of n variables. Let d be the degree of ideal J. T.W. Dubè has proved in [48] that the degrees of the elements of a reduced Gröbner basis are always bounded by the expression $2(\frac{1}{2}d^2 + d)^{2^{n-1}}$. This polynomial is Gröbner Basis computation is doubly-exponential in n and polynomial in the degree d. [49].

In this thesis, we work with zero-dimensional ideals, which have a finite number of solutions. In [49], it is shown that the complexity of computation of Gröbner Basis for a zero-dimensional ideal is a single-exponential bound. The complexity of SAT-solvers is similar. Gröbner Basis is known to be notoriously hard to compute in practical settings. Any efficient approach ought to overcome this complexity.

3.5 Projection of varieties and elimination ideals

To solve the problem of rectification check, we use the notion of projection of varieties to check where a circuit is rectifiable at a net. This problem if solved using Gröbner basis with elimination ideals. This sections covers some important definitions regarding those concepts.

Definition 3.24. *(Projection of variety) Given an ideal $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}[x_1, \ldots, x_n]$ and it's variety $V(J) \subset \mathbb{F}^n$, the l-th projection of V(J), denoted as $Pr_l(V(J))$, is the mapping*

$Pr_l(V(J)) : \mathbb{F}^n \to \mathbb{F}^{n-l}$, $Pr_l(a_1, \ldots, a_n) = (a_l + 1, \ldots, a_n)$

for every $a = (a_1, \ldots, a_n) \in V(J)$.

Definition 3.25. *Given an ideal $J \subset R = \mathbb{F}[x_1, \ldots, x_n]$, the l-th elimination ideal J_l is an ideal in R defined as $J_l = J \cap \mathbb{F}[x_{l+1}, \ldots, x_n]$.*

The next theorem shows how we can obtain the generators of the l-th elimination ideal using Gröbner bases.

Theorem 3.3. *(Elimination Theorem [50]). Given an ideal $J \subset R = \mathbb{F}[x_1, \ldots, x_n]$ and its Gröbner Basis G w.r.t. the lexicographical (lex) order on the variables where $x_1 > x_2 > \cdots > x_n$, then for every $0 \leq l \leq n$, we denote by G_l the Gröbner Basis of l-th elimination ideal of J and compute it as:*

$$G_l = G \cap \mathbb{F}[x_{l+1}, \ldots, x_n]$$

.

3.6 Concluding Remarks

Our procedure to compute a polynomial function that can rectify the given buggy circuit uses concepts from polynomials ideals, varieties, and ideal membership test using Gröbner Basis theory. We start by first modeling the gates in the circuit as polynomials in a ring R with coefficients from the field of rational numbers, $\mathbb{Q}$. The future chapters describe how gates are modeled as polynomials and how the concepts described in this chapter are used to compute a rectification polynomial that can rectify the given circuit.

CHAPTER 4

VERIFICATION OF INTEGER MULTIPLIERS

This chapter describes the problem of formal verification of hardware implementations of integer multipliers over the field of rational numbers, $\mathbb{Q}$, using computer algebra and algebraic geometry based approach. The paper [13] describes a model to verify integer multipliers by modeling the polynomials in $\mathbb{Q}$. This chapter describes its verification approach in detail. Given a specification polynomial f and a circuit C, it has to be proved that the circuit C correctly implements f. Otherwise, a counter example is generated that excites the bug in the design.

The arithmetic circuit is modeled as a polynomial system in $\mathbb{Q}[x_1, x_2, \ldots, x_n]$ and the verification problem is formulated as a membership test in a corresponding (radical) ideal. This requires the computation of a Gröbner basis, which is computationally expensive. To overcome this limitation, the circuit topology is analyzed, and a term order derived to represent the polynomials. In [51], it was shown that the topology of the circuit could be analyzed and the polynomials written in a way that this term order renders the set of polynomials itself a minimal Gröbner basis. In [13], the authors used a similar concept to derive such a term order. In this thesis, we use the same concept to find a similar term order that renders the polynomials itself a Gröbner Basis. Subsequently, the verification test is formulated as a much simpler case of Gröbner basis reduction via polynomial division.

4.1 Polynomial modelling for Integer multipliers

A k-bit integer multiplier is a circuit which takes as input two k-bit vectors and performs the multiplication of the integer values corresponding to the bit-vectors. Integer multipliers are extensively used in hardware design for all computations. For example, it is used in most Digital Signal Processing applications.

Verification of integer multipliers is a valid problem in computer algebra. In [52], the problem of verification of integer circuits was defined over finite integer rings $\mathbb{Z}_{2^k}$. However, this model is not efficient for integer arithmetic circuits as the set of integers is not a field. Many techniques that are valid over fields do not work over the set of integers. This complicates the problem of verification. Many concepts, such as Nullstellenstaz results, which are used in the formulations shown in this thesis, are valid only for fields. Computation of Gröbner Basis and the division algorithms become more complicated to use in rings. Over integer rings, it is required to use the generalization of the Buchburger's algorithm, the Möller's algorithm, explained in [26]. This algorithm is extremely complicated, which makes it impractical for the use of verification.

To overcome this limitation, the authors in [13] have developed a sound and complete model for verification of integer multipliers. This model solves the problem of verification over the field of rationals $\mathbb{Q}$. Since the set of rational numbers $\mathbb{Q}$ is a field and $\mathbb{Z} \subset \mathbb{Q}$, many techniques unavailable over integer rings can be used to efficiently solve the problem of verification. In fact, the theory in [13] is such that the circuit can be modeled over any field $\mathbb{F}$ that contains the set of integers, $\mathbb{Z} \subset \mathbb{F}$.

The material in this chapter is borrowed from [13]. The notation has been changed to fit our application. By building on the theory in this chapter, we describe the procedure to compute a rectification polynomial which can be synthesized as a sub-circuit.

4.2 Polynomial modeling

Let $R = \mathbb{Q}[x_1, \ldots, x_n]$, and let f be a polynomial in R that acts as specification for a given circuit. Note that f is a polynomial with integer coefficients. Consider circuit C with two k-bit vectors as inputs. Let input vectors be $a_0, \ldots, a_{k-1}$ and $b_0, \ldots, b_{k-1}$. The output of the circuit C is a $2k$-bit vector $s_0, \ldots, s_{2k-1}$.

Specification: Given two k-bit vectors, the specification can be modeled as polynomial in $\mathbb{Q}$ as:

$$f = \sum_{i=0}^{2k-1} 2^i s_i - \sum_{i=0}^{k-1} 2^i a_i \cdot \sum_{i=0}^{k-1} 2^i b_i \tag{4.1}$$

where $a_i, b_i, s_i \in \{0, 1\} \subset \mathbb{Q}$.

Implementation: Given a gate-level circuit netlist, we map the gate-level Boolean operators (AND, OR, NOT, XOR) to polynomials over $\mathbb{Q}$ using the following one-to-one mapping:

$$\begin{aligned} u = \neg v &\implies 0 = u - 1 + v \\ u = v \wedge w &\implies 0 = u - vw \\ u = v \vee w &\implies 0 = u - v - w + vw \\ u = v \oplus w &\implies 0 = u - v - w + 2vw \end{aligned} \tag{4.2}$$

Example 4.1. *Consider the equation with Boolean operators:*

$$z = a \oplus (b \wedge c).$$

The equation modeled over $\mathbb{Q}$ *is:*

$$z - a - b \cdot c + 2 \cdot a \cdot b \cdot c = 0$$

The left-hand side expression is a polynomial in $\mathbb{Q}[a, b, c, z]$*:*

$$z - a - b \cdot c + 2 \cdot a \cdot b \cdot c$$

Therefore, we can transform the entire circuit implementation as a system of polynomials over $\mathbb{Q}$ as follows:

$$\left.\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ &\vdots \\ f_s(x_1, x_2, \cdots, x_n) &= 0 \end{aligned}\right\} \quad \textit{Bit-level circuit constraints} \tag{4.3}$$

Example 4.2. *Consider a 3-bit integer multiplier shown in Fig. 4.1. The variables a_0, a_1, a_2, b_0, b_1, b_2 are primary inputs, s_0, s_1, s_2, s_3, s_4, s_5 are primary outputs. The specification polynomial for this circuit is:*

$$f = -(1 \cdot s_0 + 2 \cdot s_1 + 4 \cdot s_2 + 8 \cdot s_3 + 16 \cdot s_4 + 32 \cdot s_5) + (1 \cdot a_0 + 2 \cdot a_1 + 4 \cdot a_2) \cdot (1 \cdot b_0 + 2 \cdot b_1 + 4 \cdot b_2)$$

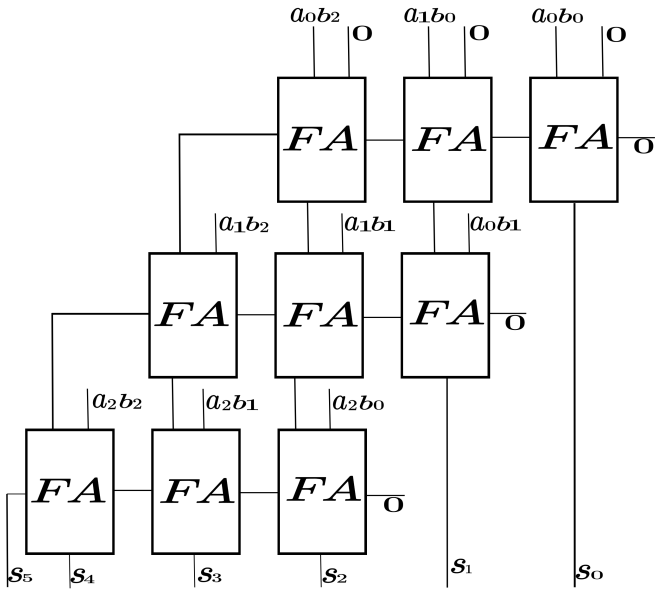

Figure 4.1: A 3-bit integer multiplier

Using the mapping rules in Eqn. (4.2), we can obtain a system of equations for the circuit in Fig. 4.1.

$$\left.\begin{aligned} f_0 &= m_0 - a_0 \cdot b_0 \\ f_1 &= m_1 + 2 \cdot n_{14} \cdot n_{15} - n_{14} - n_{15} \\ f_2 &= m_2 + 2 \cdot n_{17} \cdot n_{22} - n_{17} - n_{22} \\ f_3 &= m_3 + 2 \cdot n_{30} \cdot n_{31} - n_{30} - n_{31} \\ f_4 &= m_4 + 2 \cdot n_{35} \cdot n_{38} - n_{35} - n_{38} \\ f_5 &= m_5 + n_{40} \cdot n_{41} - n_{40} - n_{41} \end{aligned}\right\} \quad \textit{Bit-level circuit constraints}$$

$$
\left.
\begin{aligned}
f_6 &= n_{40} - n_{35} \cdot n_{38} \\
f_7 &= n_{35} + 2 \cdot n_{33} \cdot n_{34} - n_{33} - n_{34} \\
f_8 &= n_{38} - n_{36} \cdot n_{37} \\
f_9 &= n_{36} - n_{30} \cdot n_{31} \\
f_{10} &= n_{30} + 2 \cdot n_{24} \cdot n_{29} - n_{24} - n_{29} \\
f_{11} &= n_{31} - n_{17} \cdot n_{22} \\
f_{12} &= n_{17} - a_0 \cdot b_2 \\
f_{13} &= n_{22} + 2 \cdot n_{20} \cdot n_{21} - n_{20} - n_{21} \\
f_{14} &= n_{37} - n_{24} \cdot n_{29} \\
f_{15} &= n_{24} - a_1 \cdot b_2 \\
f_{16} &= n_{29} + 2 \cdot n_{25} \cdot n_{28} - n_{25} - n_{28} \\
f_{17} &= n_{41} - n_{33} \cdot n_{34} \\
f_{18} &= n_{33} - a_2 \cdot b_2 \\
f_{19} &= n_{34} - n_{25} \cdot n_{28} \\
f_{20} &= n_{25} - a_2 \cdot b_1 \\
f_{21} &= n_{28} + n_{26} \cdot n_{27} - n_{26} - n_{27} \\
f_{22} &= n_{26} - n_{20} \cdot n_{21} \\
f_{23} &= n_{20} + 2 \cdot n_{18} \cdot n_{19} - n_{18} - n_{19} \\
f_{24} &= n_{21} - n_{14} \cdot n_{15} \\
f_{25} &= n_{14} - a_0 \cdot b_1 \\
f_{26} &= n_{15} - a_1 \cdot b_0 \\
f_{27} &= n_{27} - n_{18} \cdot n_{19} \\
f_{28} &= n_{18} - a_1 \cdot b_1 \\
f_{29} &= n_{19} - a_2 \cdot b_0
\end{aligned}
\right\}
\text{\textit{Bit-level circuit constraints}}
$$

4.3 Verification formulation as Ideal Membership test

To formulate the verification test, we first analyze the circuit and model the Boolean gate-level operators as polynomials over $\mathbb{Q}$, as given by the mappings of Eqn. (4.2). Let $\{f_1, f_2, \ldots, f_s\}$ denote this set of polynomials derived from the implementation. Let $\{x_1, x_2, \ldots, x_n\}$ denote all the variables in the polynomial system, which are the nets in the circuit implementation. As a consequence, $\{f_1, f_2, \ldots, f_s\} \in \mathbb{Q}[x_1, \ldots, x_n]$. Let $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{Q}[x_1, \ldots, x_n]$ denote the ideal generated by these polynomials.

To prove that the specification polynomial (f) matches the implementation $(J = \langle f_1, \ldots, f_s \rangle)$, we need to check whether f agrees with all the solutions of J over the field $\overline{\mathbb{Q}}$. That is, we need to check *whether or not f vanishes on the variety* $V(J)$. This is because for all points (solutions) $p \in V(J)$, if $f(p) = 0$, then the implementation matches the specification. And if $f(p) \neq 0$ for any point p, then p corresponds to a bug in the design.

Now if f vanishes on $V(J)$, according to Proposition 3.2, we know that f should be a member of the radical ideal $I(V(J))$. Therefore, our verification test can be modeled as membership testing of f in the (radical) ideal $I(V(J))$. To solve this problem, we need to first derive the generators of $I(V(J))$ (note that we are only given the generators of J), and then perform the ideal membership test using the Gröbner basis algorithm.

4.3.1 Generating $I(V(J))$

Strong Nullstellensatz establishes correspondences between ideals and their radicals. As given in Theorem 3.1, $I(V_{\overline{\mathbb{F}}}(J)) = \sqrt{J}$, where the variety V is taken over the algebraically closed field $\overline{\mathbb{F}}$. The field of rational numbers are not algebraically closed. However, it can be proved that $I(V(J)) = J + J_0$. The proof for this result over finite fields is given in [53]. The same results also extends to verification of integer multipliers over $\mathbb{Q}$ shown in There results are derived in more detail in this section.

Consider the polynomial $x_l^2 - x_l$. This polynomial vanishes on $\{0, 1\}$ over the field $\overline{\mathbb{Q}}$, i.e. $V_{\overline{\mathbb{Q}}}(x_l^2 - x_l) = \{0, 1\}$. Over multivariate polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$,

$V_{\overline{\mathbb{Q}}}(x_l^2 - x_l : l = 1, \ldots, n) = \{0,1\}^n$.

Let ideal $J = \langle f_1, \ldots, f_s \rangle$ and ideal $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$.

Lemma 4.1. *Let $J \subset \mathbb{Q}[x_1, \ldots, x_n]$ be any ideal and let $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Then $V_{\mathbb{Q}}(J) = V_{\overline{\mathbb{Q}}}(J + J_0)$.*

As a consequence of the above lemma, variety of any ideal J over $\mathbb{Q}$ can be equivalently analyzed over its algebraic closure $\overline{\mathbb{Q}}$ by just appending to J all the polynomials J_0. These polynomials do not change the zero-set of J but allow the same analysis over the algebraic closure.

Another consequence of the above lemma is that since $V(J_0) = \{0,1\}^n$, we can say that $V(J + J_0) \subseteq \{0,1\}^n$. Thus we can restrict the solutions to Boolean values $\{0,1\}$.

Lemma 4.2. $I(V_{\mathbb{Q}}(J)) = I(V_{\overline{\mathbb{Q}}}(J + J_0)) = \sqrt{J + J_0}$.

Proof. As shown above, $V_{\mathbb{Q}}(J) = V_{\overline{\mathbb{Q}}}(J + J_0)$. Therefore, $I(V_{\mathbb{Q}}(J)) = I(V_{\overline{\mathbb{Q}}}(J + J_0))$. According to Strong Nullstellensatz, $I(V_{\overline{\mathbb{Q}}}(J + J_0)) = \sqrt{J + J_0}$. Thus:

$$I(V_{\mathbb{Q}}(J)) = I(V_{\overline{\mathbb{Q}}}(J + J_0)) = \sqrt{J + J_0}. \tag{4.4}$$

■

Lemma 4.3. *Let J be any arbitrary polynomial ideal in $\mathbb{Q}[x_1, \ldots, x_n]$ and $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Then $J + J_0$ is radical. In other words, $\sqrt{J + J_0} = J + J_0$.*

Putting together the above results, we finally arrive at the following application of Nullstellensatz over rational number.

Theorem 4.1. *Let J be any arbitrary polynomial ideal in $\mathbb{Q}[x_1, \ldots, x_n]$ and $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Then,*

$$I(V_{\mathbb{Q}}(J)) = I(V_{\overline{\mathbb{Q}}}(J + J_0)) = \sqrt{J + J_0} = J + J_0. \tag{4.5}$$

Overall verification formulation: Through Strong Nullstellensatz in Theorem. 4.1, given an ideal J, we can directly construct ideal $I(V(J)) = J + J_0$. For our verification problem, we take the polynomials $\{f_1, \ldots, f_s\}$ representing the circuit

constraints to generate ideal J. Then we append the ideal $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Verification problem can now be formulated as ideal membership of f in $J + J_0$. If $f \in (J + J_0)$, correctness of the circuit is established. Otherwise, there is a bug in the design. To test if $f \in (J + J_0)$, it is required to compute a Gröbner basis G of the ideal $J + J_0$. Then, we reduce f w.r.t. G: i.e., $f \xrightarrow{G}_+ r$. If $r = 0$, then the circuit is correct, otherwise there is a bug in the design.

While this approach seems reasonably simple, the complexity of Gröbner basis computation can make verification infeasible. In section 3.4, we discuss the complexity of Gröbner Basis computation. We can overcome this complexity by deriving a term order based on the circuit topology which makes the set of polynomials $\{f_1, \ldots, f_s, x_1^2 - x_1, \ldots, x_n^2 - x_n\}$ itself a Gröbner basis of $J + J_0$ obviating the need to apply Buchberger's algorithm.

4.4 Overcoming complexity of Gröbner Basis computation

Just as variable orderings play a critical role in constructing BDDs and solving SAT feasibly, the Gröbner basis computation is also highly susceptible to the term orderings imposed on the polynomials. Therefore a key step to improve/avoid the high complexity of Gröbner basis computation is to derive a "good" term order.

Buchbergers work initially laid the foundation for computing Gröbner bases. Subsequently, many improvements were introduced to improve the efficiency of Buchbergers algorithm. Two of the most important improvements are the chain and product criteria. For our particular circuit verification application, we exploit the product criteria.

Lemma 4.4. *[Product Criterion [54]] Let $\mathbb{F}$ be any field, and $f, g \in \mathbb{F}[x_1, \cdots, x_d]$ be polynomials. If the equality $lm(f) \cdot lm(g) = LCM(lm(f), lm(g))$ holds, then*

$Spoly(f, g) \xrightarrow{G}_+ 0$.

The above result states that when the leading monomials of f, g are relatively prime, then $Spoly(f, g)$ always reduces to 0 modulo G. Thus $Spoly(f, g)$ need not be considered in Buchbergers algorithm. Modern computer algebra engines perform this check to avoid unnecessary $Spoly(f, g)$ computations. If we could

analyze the given circuit and derive a term order such that every polynomial pair (f, g) in the generating set has relatively prime leading monomials, then for all S-polynomials, the subsequent reduction would not add any new polynomials in the basis. In other words, $Spoly(f, g) \xrightarrow{G}_{+} 0$ for all pairs f, g. Consequently, the polynomials $f_1, \ldots, f_s$ extracted from the circuit (corresponding ideal J) and represented using such a term order would itself constitute a Gröbner basis of J. In [16], the authors derive exactly such a term order, and a similar concept can be applied in our case.

Note that in our case:

- The circuit constraints $\{f_1, \ldots, f_s\}$ contain only multi-linear monomial terms;
- the output of a gate is uniquely computed, and it always appears as a "single variable term in the polynomials;
- the circuit is acyclic;

Let x_i be the output variable of any gate H_i in the circuit, and let $x_{p1}, \ldots, x_{pj}$ denote variables that are the inputs to the gate H_i. If we can represent the polynomials f_i such that $x_i >$ every monomial in the variables $x_{p1}, \ldots, x_{pj}$, then all $(f_i, f_j), i \neq j$ have relatively prime leading monomials and $\{f_1, \ldots, f_s\}$ is a Gröbner basis.

Proposition 4.1. *Let C be any arbitrary combinational circuit. Let $\{x_1, \ldots, x_n\}$ denote the set of all variables (signals) in the circuit, i.e. the primary input, intermediate and primary output variables. Perform a* ***reverse topological traversal*** *of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lexicographic term order to represent the Boolean expression for each gate as a polynomial f_i; then $f_i = x_i + tail(f_i)$. Such a term order is called* ***Reverse Topological Term Order (RTTO)****. Then the set of all polynomials $\{f_1, \ldots, f_s\}$ forms a Gröbner basis, as $lt(f_i)$ and $lt(f_j)$ for $i \neq j$ are relatively prime.*

Proposition 4.1 is proved in [55] and a similar result is described in [13]. Following Proposition 4.1, the polynomials $F = \{f_1, \ldots, f_s\}$ and $F_0 = \{x_1^2 - 1, \ldots, x_n^2 - x_n\}$ form the Gröbner Bases of ideals J and J_0 respectively.

4.5 Column-wise slicing for verification

The authors in [13] show that the circuit can be partitioned into columns and reduction for ideal membership can be done incrementally. The circuit is partitioned into different slices column-wise. The gates in within the slices are ordered topologically. The gates in each slice are modeled as a system of equations that generate an ideal. A k-bit integer multiplier is partitioned into $2k$ slices. Let those slices be $G_0, G_1, \ldots, G_{2k-1}$. Also consider the ideal $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$.

For a correct circuit, verification is done by checking if the specification polynomial is a member of the ideal generated by the circuit. The result of the computation, $f \xrightarrow{G_0, G_1, \ldots, G_{2k-1}}_{+} r$, should be $r = 0$. The algorithm used to verify multipliers in [13] is described in Algorithm 3.

Algorithm 3: Verification of integer multiplier

Input: $\{G_0, \ldots, G_{2k-1}, J_0\}$
Output: C_0
$C_{2k} = 0$;
for $i \leftarrow 2k - 1$ ***to*** *0* **do**
    $2C_{i+1} + z_i + P_i \xrightarrow{G_i + J_0} C_i$
end

For column i with $0 < i < 2k$, $P_i = \sum_{k+l=i} a_k b_l$. The circuit is verified against the specification if $C_0 = 0$. If $C_0 \neq 0$, it implies that the circuit is buggy.

Example 4.3. *Let us consider the 3-bit multiplier in Fig. 4.1 again. We impose a lexicographic term order $>$ on the variables such that $x_i > x_j$ if x_i appears earlier in reverse topological order.*

The term order imposed is:

$s_0 > s_1 > s_2 > s_3 > s_4 > s_5 > n_{40} > n_{35} > n_{38} > n_{36} > n_{30} > n_{31} > n_{17} > n_{22} > n_{37} > n_{24} > n_{29} > n_{41} > n_{33} > n_{34} > n_{25} > n_{28} > n_{26} > n_{20} > n_{21} > n_{14} > n_{15} > n_{27} > n_{18} > n_{19} > a_0 > a_1 > a_2 > b_0 > b_1 > b_2$

The gates can be partitioned into columns as shown in the figure.

$$
\begin{aligned}
G_0 &= \{m_0 - b_0 \cdot a_0\} \\
G_1 &= \{m_1 - n_{15} - n_{14} + 2 \cdot n_{15} \cdot n_{14}, \\
&\quad n_{14} - b_1 \cdot a_0, \\
&\quad n_{15} - b_0 \cdot a_1, \\
&\quad n_{21} - n_{15} \cdot n_{14}\}
\end{aligned}
$$

$$
\begin{aligned}
G_2 &= \{m_2 - n_{22} - n_{17} + 2 \cdot n_{22} \cdot n_{17}, \\
&\quad n_{22} - n_{21} - n_{20} + 2 \cdot n_{21} \cdot n_{20}, \\
&\quad n_{17} - b_2 \cdot a_0, \\
&\quad n_{20} - n_{19} - n_{18} + 2 \cdot n_{19} \cdot n_{18}, \\
&\quad n_{18} - b_1 \cdot a_1, \\
&\quad n_{27} - n_{19} \cdot n_{18}, \\
&\quad n_{28} - n_{27} - n_{26} + n_{27} \cdot n_{26}, \\
&\quad n_{26} - n_{21} \cdot n_{20}, \\
&\quad n_{31} - n_{22} \cdot n_{17}, \\
&\quad n_{19} - b_0 \cdot a_2\}
\end{aligned}
$$

$$
\begin{aligned}
G_3 &= \{m_3 - n_{31} - n_{30} + 2 \cdot n_{31} \cdot n_{30}, \\
&\quad n_{30} - n_{29} - n_{24} + 2 \cdot n_{29} \cdot n_{24}, \\
&\quad n_{38} - n_{37} - n_{36} + n_{37} \cdot n_{36}, \\
&\quad n_{36} - n_{31} \cdot n_{30}, \\
&\quad n_{24} - b_2 \cdot a_1, \\
&\quad n_{29} - n_{28} - n_{25} + 2 \cdot n_{28} \cdot n_{25}, \\
&\quad n_{25} - b_1 \cdot a_2, \\
&\quad n_{34} - n_{28} \cdot n_{25}, \\
&\quad n_{37} - n_{29} \cdot n_{24}\}
\end{aligned}
$$

$$
\begin{aligned}
G_4 = \{&m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
&n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
&n_{33} - b_2 \cdot a_2, \\
&n_{40} - n_{38} \cdot n_{35}, \\
&n_{41} - n_{34} \cdot n_{33}\} \\
G_5 = \{&m_5 - n_{41} - n_{40} + n_{41} \cdot n_{40}\}
\end{aligned}
$$

Because of the term order $>$ imposed, $G_i : 0 < i < 5$ is a Gröbner Basis itself. Algorithm 3 is applied to the above system of equations with $n = 3$ and we find that $C_0 = 0$ is computed. This indicates that the circuit is correctly implemented and it is verified.

If there is a bug in the circuit, Algorithm 3 returns a non-zero remainder, $C_0 \neq 0$. This remainder is a counter-example which excites the bug in the circuit.

Example 4.4. *Consider the polynomial equations for circuit in Fig. 4.1 that is listed in Example 4.3. Consider the polynomial $f_8 = -n_{38} + n_{37} + n_{36} - n_{37} \cdot n_{36}$, which models an OR gate at net n_{38} in the correct circuit. Suppose a bug is introduced at this net n_{38}. Let the bug be an AND gate in place of the correct OR gate. This bug means that the implementation no longer matches the specification. This bug is modeled as $f_8 = -n_{38} + n_{37} \cdot n_{36}$. Polynomials in set G_4 with the polynomial for the buggy AND gate will now be:*

$$
\begin{aligned}
G_4 = \{&m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
&n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
&\boldsymbol{n_{38} - n_{37} \cdot n_{36}}, \\
&n_{33} - b_2 \cdot a_2, \\
&n_{34} - n_{28} \cdot n_{25}, \\
&n_{36} - n_{31} \cdot n_{30}, \\
&n_{37} - n_{29} \cdot n_{24}\}
\end{aligned}
\tag{4.6}
$$

If Algorithm 3 is now applied, it returns a non-zero value for C_0.

$$C_0 =$$
$$-16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 - 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot$$
$$a_1 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2$$
$$\neq 0$$

This indicates a bug in the implementation.

4.6 Concluding Remarks

This chapter presented a detailed description of the application of algebraic geometry and the algorithm used in [13] for verification of integer multipliers. The gates of a circuit implementation are modeled as a system of equations $\{f_1, \ldots, f_s\} \subset \mathbb{Q}[x_1, \ldots, x_n]$ and verification is performed as an ideal membership test to determine if specification polynomial f is a member of the radical ideal $J + J_0 = \langle f_1, \ldots, f_s, x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. The polynomial reduction for ideal membership test is done incrementally by slicing the multiplier circuit into columns as shown in [13]. Because of the presence of bugs in the design, the verification process results in a non-zero remainder. This thesis focuses on rectification of integer arithmetic circuits. Subsequent chapters will describe how the remainder computed as the result of the verification test is used to extract information about the effect of the bug, which will assist in computing a rectification polynomial that will correct the buggy circuit.

CHAPTER 5

RECTIFICATION CHECK

Once the presence of a bug in the circuit is determined by verification, we need to attempt rectification of the circuit. In this thesis, we address only single-fix rectification. Single-fix rectification means that irrespective of the type and number of bugs, the circuit can be rectified at a single net. This chapter describes the approach to obtain a subset of internal nets where single-fix rectification might be possible. Subsequently, we describe an approach to ascertain unequivocally whether a net under consideration admits single-fix rectification.

For verification, a given circuit C is modeled as a system of polynomials $F = \{f_1, \ldots, f_s\} \subset \mathbb{Q}[x_1, \ldots, x_n]$ and verified against a specification polynomial f. If there is a bug in the circuit, the ideal membership test computes a non-zero remainder. We now compute a rectification function that rectifies the circuit, so it matches the given specification. Before the computation of a rectification function is attempted, we need to find a net which admits single-fix rectification. This chapter describes the procedure to find a target net in the circuit where rectification is feasible. A circuit can always be rectified at the primary outputs, but this is equivalent to re-synthesis of the entire circuit. So it is desirable to locate candidates for rectification among internal nets. In general, every net in the circuit can be checked for rectification. But this is infeasible for large circuits. Therefore, we need to prune the search space to make the approach scalable. By examining the remainder computed in the verification process, we can obtain a subset of internal nets where single-fix rectification is likely to be possible. From this subset of nets, we need to find a net which admits single-fix rectification. This chapter describes an approach to determine if single-fix rectification can be applied at a given (target) net x_i in C.

5.1 Problem Statement

The following is our problem statement:

- Given a ring $R = \mathbb{Q}[x_1, \ldots, x_n]$, where $x_1, \ldots, x_n$ are the nets in the circuit.
- Given a word-level specification polynomial f in R. Note that f is a polynomial with integer coefficients.
- Given a buggy k-bit integer multiplier circuit C. The bit-level primary inputs are $\{a_0, \ldots, a_{k-1}\}$ and $\{b_0, \ldots, b_{k-1}\}$. The bit-level primary outputs are $\{z_0, \ldots, z_{2k-1}\}$.
- Represent the gates of the circuit C using a system of polynomials as shown in Eqn. (4.2). The set of polynomials modeling the circuit are $F = \{f_1, \ldots, f_s\} \subset R = \mathbb{Q}[x_1, \ldots, x_n]$. Generate the set of vanishing polynomials, $F_0 = \{x_1^2 - x_1, \ldots, x_n^2 - x_n\}$.
- Perform topological analysis of the circuit and obtain RTTO.
- Compute the non-zero remainder r through the ideal membership testing for verification, $f \xrightarrow{F \cup F_0}_{+} r$.

Our goal is find a subset of internal nets in the circuit C, $\mathcal{N} \subseteq \{x_1, \ldots, x_n\}$, as potentially rectifiable target nets. Once this subset of nets is found, we identify a target net $x_i \subseteq \mathcal{N}$, which admits single-fix rectification.

Let $X_{PI} \subset \{x_1, \ldots, x_n\}$ be the primary input variables of the circuit.

Definition 5.1. *Single fix-rectification at target net x_i means that there exists a polynomial function $U(X_{PI})$ which, when implemented at net x_i, ensures that circuit C correctly implements the specification f.*

In Def. 5.1, $U(X_{PI})$ is the rectification polynomial in primary input variables only. We prove that this polynomial is a mapping from $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$. This property means that we can compute $U(X_{PI}) \pmod 2$ and still obtain the same function. Since addition and multiplication $\pmod 2$ can be interpreted as XOR

and AND operations respectively, we can synthesize this polynomial as a logic circuit.

Example 5.1. *Consider a buggy 2-bit multiplier in Fig. 5.1.*

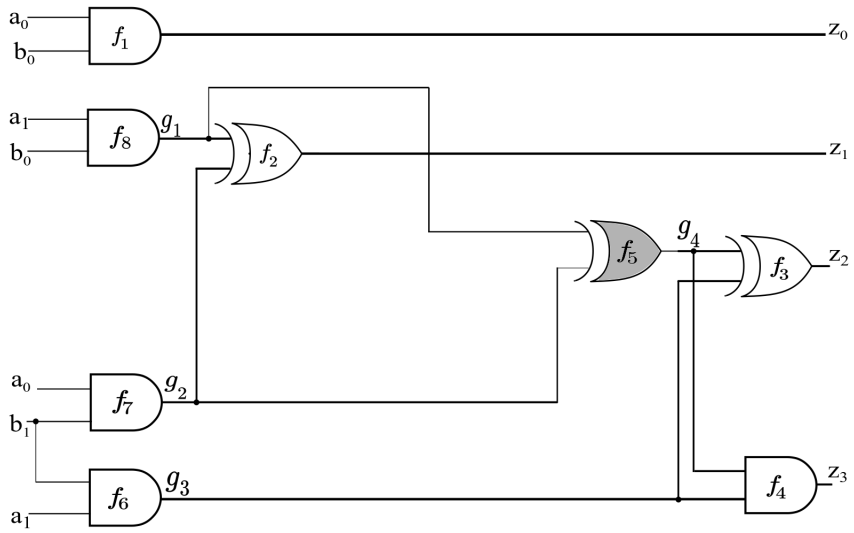

Figure 5.1: 2-bit Integer multiplier with a bug. AND gate at g_4 replaced by an XOR gate.

- *Given the ring* $R = \mathbb{Q}[z_0, z_1, z_2, z_3, g_4, g_3, g_2, g_1, a_0, a_1, b_0, b_1]$.
- *Given the specification polynomial*
  $f = -(8 \cdot z_3 + 4 \cdot z_2 + 2 \cdot z_1 + z_0) + (2 \cdot a_1 + a_0) \cdot (2 \cdot b_1 + b_0)$
- *Given a 2-bit integer multiplier circuit C. The bit-level primary inputs are* $\{a_0, \ldots, a_1\}$ *and* $\{b_0, \ldots, b_1\}$. *The bit-level primary outputs are* $\{z_0, \ldots, z_3\}$.
- *A bug is introduced by replacing the correct AND gate at* f_6 *by an XOR gate.*
- *Model the gates of the circuit as a system of polynomials as shown in Eqn. (4.2),* $F = \{f_1, \ldots, f_8\}$. *Generate the set of vanishing polynomials,* $F_0 = \{a_0^2 - a_0, \ldots, z_3^2 - z_3\}$.
- *Analyze the topology of the circuit and derive RTTO. The term order imposed is:* $lex : \{z_0 > z_1 > z_2 > z_3\} > \{g_4\} > \{g_3 > g_2 > g_1\} > \{a_0 > a_1 > b_0 > b_1\}$

- *Given the non-zero remainder r computed during the ideal membership testing for verification,* $f \xrightarrow{F \cup F_0}_+ r$.
  $r = 12 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 - 4 \cdot a_0 \cdot b_1 - 4 \cdot a_1 \cdot b_0$

A subset of internal nets $\mathcal{N} = \{g_4, g_3, g_2, g_1\}$ *is identified as as a set of potentially rectifiable nets. From this subset of nets, we find* $x_i = g_4$ *as a net which admits single-fix rectification.*

In subsequent sections, we describe how to identify the subset $\mathcal{N}$ and the approach to select x_i.

5.2 Identifying potentially rectifiable target nets

When ideal membership test for verification is performed, a non-zero remainder r is computed for a buggy circuit. This remainder r is used to prune the search space and identify target nets which may admit single-fix rectification. This approach is inspired by the procedure used on finite field arithmetic circuits described in [46].

A set of nets where rectification is possible is identified by first determining the primary output bits where the bug is observable. When a bug affects multiple outputs, a single-fix rectification might exist only at the nets that lie in the intersection of the respective fanin-cones of the affected outputs. These affected outputs are identified by examining the coefficients of the remainder r.

Proposition 5.1. *The circuit C has k-bit operands, and the specification polynomial is expressed as*

$$f = \sum_{i=0}^{2k-1} 2^i z_i - \sum_{i=0}^{k-1} 2^i a_i \cdot \sum_{i=0}^{k-1} 2^i b_i$$

Then the non-zero remainder r computed through the verification test can be partitioned based on the coefficients of the monomials in r and re-expressed as:

$$r = 2^0(r_0) + 2^1(r_1) + \cdots + 2^{2k-1}(r_{2k-1})$$

Non-zero terms r_i *(with coefficient* 2^i*) imply that the effect of the bug is observable at the bit-level outputs* z_i.

When we have identified the output bits where the effect of the bug is observable, we consider the transitive fanin cones of logic of the output bits z_i. The intersection of the respective fanin-cones of the affected outputs is the set of potentially rectifiable target nets $\mathcal{N} \subseteq [x_1, \ldots, x_n]$.

Potential rectifiable nets: The nets in the intersection of the fanin cones of the outputs impacted by the bug comprise the set of potentially rectifiable target nets. Usually, the effect of the bugs on the primary outputs can also be identified using random simulation. However, since we have a remainder computed during verification which contains this information, we present a sound approach to identify potentially rectifiable target nets for single-fix rectification.

Example 5.2. *Consider again the circuit in Fig. 5.1 from Example 5.1. The non-zero remainder r computed during verification test is:*

$$r = 12 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 - 4 \cdot a_0 \cdot b_1 - 4 \cdot a_1 \cdot b_0$$

This remainder r can be re-expressed as:

$$\begin{aligned} r &= 12 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 - 4 \cdot a_0 \cdot b_1 - 4 \cdot a_1 \cdot b_0 \\ &= 8 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 + 4 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 - 4 \cdot a_0 \cdot b_1 - 4 \cdot a_1 \cdot b_0 \\ &= \mathbf{2^3}(a_0 \cdot a_1 \cdot b_0 \cdot b_1 - 4 \cdot a_0 \cdot b_1) + \mathbf{2^2}(a_0 \cdot a_1 \cdot b_0 \cdot b_1 - a_0 \cdot b_1 - 4 \cdot a_1 \cdot b_0) \end{aligned}$$

From the remainder, we ascertain that the bug is observable at z_3 and z_2. The intersection of the fanin cones of logic of z_3 and z_2 give the set $\mathcal{N} = \{g_4, g_3, g_2, g_1\}$.

For integer multiplier circuits, we use the model described in [13] for verification. A k-bit multiplier is divided into $2k$ slices column-wise, based on the structure of the circuit. Let the gates in the circuit be modeled as polynomials in sets $G_0, \ldots, G_{2k-1}$. The following example shows how our approach identifies the set of potentially rectifiable nets are identified using the verification model of [13].

Example 5.3. *Recall the circuit in Fig. 4.1 of a 3-bit integer multiplier. In Ex. 4.4, we introduce a bug in this circuit and perform verification. The bug introduced was a correct*

OR gate replaced by an AND gate. The remainder computed from the verification test for the circuit with this bug is:

$$\begin{aligned} r = &-16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 - 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\ &+ 16 \cdot a_0 \cdot a_1 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\ &+ 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \end{aligned}$$

Based on the coefficients of the polynomial r, only a single output s_4 is affected. We now consider the fanin cone of logic of the output s_4 that lie in slices G_4, G_3, G_2, G_1. These nets are

$$\begin{aligned} \mathcal{N} = \{&n_{19}, n_{18}, n_{27}, n_{15}, n_{14}, n_{21}, n_{20}, n_{26}, n_{28}, n_{25}, n_{34}, \\ &n_{33}, n_{29}, n_{24}, n_{37}, n_{22}, n_{17}, n_{31}, n_{30}, n_{36}, n_{38}, n_{35}, m_4\} \end{aligned}$$

These nets are sufficient to test for single-fix rectifiablity.

Once a subset of nets where single-fix rectification is possible, we need to ascertain if a net under consideration admits single-fix rectification or not.

5.3 Rectification check

After post-verification debugging is performed to identify a set of nets $\mathcal{N} \subseteq \{x_1, \ldots, x_n\}$ that are potentially rectification target nets, we now present an approach that *confirms* whether or not the circuit can indeed be single-fix-rectified at net x_i. We state and prove the rectification theorem that checks for the existence of $U(X_{PI})$ as a single-fix rectification function at x_i. In the following theorem, we use the notation $V(J)|_{X_{PI}}$ to denote the projection of variety on primary inputs only.

Theorem 5.1. *Let $F = \{f_1, \ldots, f_s\}$ be the set of polynomials that models the given circuit C. Let ideal $J = \langle F \rangle \subset \mathbb{Q}[x_1, \ldots, x_n]$. Let set $F_0 = \{x_l^2 - x_l : x_l \in X_{PI}\}$ and ideal $J_0^{PI} = \langle F_0 \rangle$. Construct two ideals:*

$$\begin{aligned} J_L &= \langle f_1, \ldots, f_{i-1}, f_i : x_i - 1, f_{i+1}, \ldots, f_s \rangle \\ J_H &= \langle f_1, \ldots, f_{i-1}, f_i : x_i - 0, f_{i+1}, \ldots, f_s \rangle \end{aligned} \tag{5.1}$$

where the polynomials $\{f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_s\}$ are the same as in the generators of ideal J (representing the circuit), and f_i is replaced with $f_i = x_i - 1$ in J_L and $f_i = x_i - 0$ in J_H, respectively. Perform the reductions:

$$\begin{aligned} f &\xrightarrow{J_L+J_0}_+ r_L \\ f &\xrightarrow{J_H+J_0}_+ r_H \end{aligned} \tag{5.2}$$

The buggy circuit C admits single-fix rectification at net x_i if and only if

$$V_{\overline{\mathbb{Q}}}(\langle r_L \rangle + J_0)|_{X_{PI}} \cup V_{\overline{\mathbb{Q}}}(\langle r_H \rangle + J_0)|_{X_{PI}} = \{0,1\}^{|x_{PI}|} \tag{5.3}$$

Proof. As rectification at net x_i makes the circuit C match the specification f, f should vanish on $V(J)$. Thus, the rectification condition can be equivalently stated as: "f vanishes on $V(J) \iff V_{\overline{\mathbb{Q}}}(\langle r_L \rangle + J_0) \cup V_{\overline{\mathbb{Q}}}(\langle r_H \rangle + J_0) = \{0,1\}^{|x_{PI}|} = V(J_0^{PI})$".

(i) To prove $\Rightarrow$: Let $x_{PI} \in \{0,1\}^{X_{PI}}$ be an assignment to the primary input variables of C. For every point x_{PI}, there exists a corresponding assignment x_{int} to the rest of the variables of the circuit. For each primary input assignment, the target net x_i evaluates to either $x_i = 0$ or $x_i = 1$. When $x_i = 0$, then J_H vanishes on the point (x_{PI}, x_{int}). Likewise, when $x_i = 1$, J_L vanishes on (x_{PI}, x_{int}). Since $f \xrightarrow{J_H+J_0}_+ r_H$ and $f \xrightarrow{J_L,J_0}_+ r_L$, and f vanishes on the point (x_{PI}, x_{int}), we obtain that either $r_H(x_{PI}) = 0$ or $r_L(x_{PI}) = 0$. In other words, for every primary input assignment x_{PI}, either r_L or r_H vanish. This implies that $V_{\overline{\mathbb{Q}}}(\langle r_L \rangle + J_0) \cup V_{\overline{\mathbb{Q}}}(\langle r_H \rangle + J_0) = \{0,1\}^{|x_{PI}|} = V(J_0^{PI})$.

(ii) To prove $\Leftarrow$: Say there exists an assignment to the primary inputs $x_{PI} \in \{0,1\}^{|x_{PI}|}$ such that r_H vanishes on x_{PI}, i.e. $r_H(x_{PI}) = 0$. Corresponding to x_{PI}, there exists an assignment to the rest of the variables of the circuit x_{int}. As $f \xrightarrow{J_H+J_0}_+ r_H$, we have that f is a member of the ideal $J_H + J_0 + \langle r_H \rangle$. Therefore, when $r_H(x_{PI}) = 0$, the ideal J_H also vanishes on (x_{PI}, x_{int}), and J_0 by definition vanishes on $\{0,1\}^{X_{PI}}$. This implies that $f(x_{PI}, x_{int}) = 0$. Similarly, the argument also holds

that when $r_L(x_{PI}) = 0$, then $f(x_{PI}, x_{int}) = 0$. This proves that for all primary inputs if r_L and r_H vanish, then f vanishes too; and that completes the proof.

■

Let x_i be the net under consideration and ideals J_L and J_H are constructed. If Theorem 5.1 is satisfied, the proof for the theorem implies that there exists a polynomial function at x_i, which maps from $\{0,1\}^{|X_{PI}|}$ to $\{0,1\}$. This polynomial function (truth table) implemented at net x_i corrects the circuit so it matches the specification. This rectification check can also be performed using the column-wise model used in [13]. We know that $\langle G_0 + \cdots + G_{2k-1} \rangle = \langle J \rangle$, where G_i is the Gröbner Bases of the ideal generated by the polynomials modeling the gates in column i. Let the polynomial $f_i : x_i + tail(f_i)$ be in set G_j. Therefore, if we construct two ideals G_{j_L} and G_{j_H} according to Eqn. (5.1), then $\langle G_0 + \cdots + G_{j_L} + \cdots + G_{2k-1} \rangle = \langle J_L \rangle$ and $\langle G_0 + \cdots + G_{j_H} + \cdots + G_{2k-1} \rangle = \langle J_H \rangle$.

Computation: As described in Def. 3.16, union of varieties is equivalent to product of ideals. To compute the union of varieties in Eqn. 5.3 for the rectification check, we compute Gröbner Basis $G = GB((r_L \cdot r_H) + J_0)$ and check if G constitutes only vanishing polynomials.

Example 5.4. *Example 4.3 shows the system of equations modeling the correct implementation of the 3-bit integer multiplier shown in Fig. 4.1. In Ex.* 4.4*, a bug is introduced in this circuit at net n_{38}. The correct OR gate at n_{38} was replaced by an AND gate. This changed the set G_4 to the following:*

$$
\begin{aligned}
G_4 = \{&m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
&n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
&\boldsymbol{n_{38} - n_{37} \cdot n_{36}}, \\
&n_{33} - b_2 \cdot a_2, \\
&n_{34} - n_{28} \cdot n_{25}, \\
&n_{36} - n_{31} \cdot n_{30}, \\
&n_{37} - n_{29} \cdot n_{24}\}
\end{aligned}
\tag{5.4}
$$

This buggy circuit was verified against the specification and a non-zero remainder was computed. In Ex. 5.3, we found a subset of internal nets where single-fix rectification might be possible.

$$
\begin{aligned}
\mathcal{N} = \{&n_{19}, n_{18}, n_{27}, n_{15}, n_{14}, n_{21}, n_{20}, n_{26}, n_{28}, n_{25}, n_{34}, \\
&n_{33}, n_{29}, n_{24}, n_{37}, n_{22}, n_{17}, n_{31}, n_{30}, n_{36}, n_{38}, n_{35}, m_4\}
\end{aligned}
$$

From this set of nets $\mathcal{N}$, we need to confirm which net admits single-fix rectification. As the bug was introduced at n_{38}, it is obvious that the circuit is rectifiable at n_{38}.

Let us construct two ideals as shown in Eqn. 5.1:

$$
\begin{aligned}
G_{4_L} = \langle &m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
&n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
&\boldsymbol{n_{38} - 1}, \\
&n_{33} - b_2 \cdot a_2, \\
&n_{34} - n_{28} \cdot n_{25}, \\
&n_{36} - n_{31} \cdot n_{30}, \\
&n_{37} - n_{29} \cdot n_{24}\rangle
\end{aligned}
\tag{5.5}
$$

$$
\begin{aligned}
G_{4_H} = \langle & m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
& n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
& \boldsymbol{n_{38} - 0}, \\
& n_{33} - b_2 \cdot a_2, \\
& n_{34} - n_{28} \cdot n_{25}, \\
& n_{36} - n_{31} \cdot n_{30}, \\
& n_{37} - n_{29} \cdot n_{24} \rangle
\end{aligned}
\tag{5.6}
$$

Now we perform the reductions as shown in Eqn. 5.2:

$$
\begin{aligned}
f &\xrightarrow{G_0+\cdots+G_{4_L}+G_5+J_0} r_L \\
f &\xrightarrow{G_0+\cdots+G_{4_H}+G_5+J_0} r_H
\end{aligned}
$$

using Algorithm 3. We compute:

$$
\begin{aligned}
r_L = {} & 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\
& - 16 \cdot a_0 \cdot a_1 \cdot b_1 \cdot b_2 - 16 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\
& - 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 + 16 \\
r_H = {} & 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\
& - 16 \cdot a_0 \cdot a_1 \cdot b_1 \cdot b_2 - 16 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\
& - 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2
\end{aligned}
$$

We compute $G = GB((r_L \cdot r_H) + J_0)$ and we find G constitutes only the vanishing polynomials. Therefore, the circuit can be rectified at n_{38}.

We also apply Theorem 5.1 to ascertain if a net does not allow single-fix rectification.

Example 5.5. *Example 4.3 shows the system of equations modeling the correct implementation of the 3-bit integer multiplier shown in Fig. 4.1. In Ex.* 4.4*, a bug is introduced*

in this circuit at net n_{38}. The correct OR gate at n_{38} was replaced by an AND gate. This changed the set G_4 to the following:

$$\begin{aligned} G_4 = \{&m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\ &n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\ &\boldsymbol{n_{38} - n_{37} \cdot n_{36}}, \\ &n_{33} - b_2 \cdot a_2, \\ &n_{34} - n_{28} \cdot n_{25}, \\ &n_{36} - n_{31} \cdot n_{30}, \\ &n_{37} - n_{29} \cdot n_{24}\} \end{aligned} \tag{5.7}$$

This buggy circuit was verified against the specification and a non-zero remainder was computed. In Ex. 5.3, we found a subset of internal nets where single-fix rectification might be possible.

$$\begin{aligned} \mathcal{N} = \{&n_{19}, n_{18}, n_{27}, n_{15}, n_{14}, n_{21}, n_{20}, n_{26}, n_{28}, n_{25}, n_{34}, \\ &n_{33}, n_{29}, n_{24}, n_{37}, n_{22}, n_{17}, n_{31}, n_{30}, n_{36}, n_{38}, n_{35}, m_4\} \end{aligned}$$

From this set of nets $\mathcal{N}$, we need to confirm which net admits single-fix rectification. Let us try to rectify the bug at net n_{37}.

Let us construct two ideals as shown in Eqn. 5.1:

$$\begin{aligned} G_{4_L} = \langle &m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\ &n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\ &n_{38} - n_{37} \cdot n_{36}, \\ &n_{33} - b_2 \cdot a_2, \\ &n_{34} - n_{28} \cdot n_{25}, \\ &n_{36} - n_{31} \cdot n_{30}, \\ &\boldsymbol{n_{37} - 1}\rangle \end{aligned} \tag{5.8}$$

$$
\begin{aligned}
G_{4_H} = \langle & m_4 - n_{38} - n_{35} + 2 \cdot n_{38} \cdot n_{35}, \\
& n_{35} - n_{34} - n_{33} + 2 \cdot n_{34} \cdot n_{33}, \\
& n_{38} - n_{37} \cdot n_{36}, \\
& n_{33} - b_2 \cdot a_2, \\
& n_{34} - n_{28} \cdot n_{25}, \\
& n_{36} - n_{31} \cdot n_{30}, \\
& \boldsymbol{n_{37} - 0} \rangle
\end{aligned}
\tag{5.9}
$$

Now we perform the reductions as shown in Eqn. 5.2:

$$f \xrightarrow{G_0+\cdots+G_{4_L}+G_5+J_0} r_L$$

$$f \xrightarrow{G_0+\cdots+G_{4_H}+G_5+J_0} r_H$$

using Algorithm 3. We compute:

$$
\begin{aligned}
r_L = & -16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\
& + 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\
r_H = & -16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 - 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\
& + 16 \cdot a_0 \cdot a_1 \cdot b_1 \cdot b_2 + 16 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - 16 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\
& + 16 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2
\end{aligned}
$$

We compute $G = GB((r_L \cdot r_H) + J_0)$ and we find G does not constitute the vanishing polynomials. Therefore, the circuit cannot be rectified at n_{37}.

Therefore, there exists a polynomial function mapping from $\{0,1\}^{|X_{PI}|}$ to $\{0,1\}$ that can be implemented at n_{38} such that the circuit is rectified and correctly implements the specification.

5.4 Concluding remarks

Single-fix rectification means the buggy circuit can be rectified at a single location (net), irrespective of the type and number of bugs. The presence of a bug in a circuit is identified when the ideal membership testing computes a non-zero remainder. By examining the coefficients of the remainder, we can identify the outputs where the bug is observable. By intersecting the fanin cones of logic of the affected outputs, we can obtain a subset of internal nets where single-fix rectification might be possible. From this subset, we find a net x_i where we have confirmed that rectification is possible by applying the rectification check theorem. Once we know net x_i admits single-fix rectification, we attempt to compute a rectification polynomial. The next chapter describes the procedure used to compute a rectification polynomial which corrects the buggy circuit.

CHAPTER 6

COMPUTING RECTIFICATION FUNCTION

The presence of a bug in a circuit is determined by verification. When the bug is detected, rectification is attempted to correct the circuit. Using the rectification check theorem described in Chapter 5, we can find an internal net in the circuit which admits single-fix rectification. This means that there exists a polynomial function which can be implemented at that net such that the circuit is rectified and it exactly matches the given specification. This chapter describes the method to compute one such a rectification function using extended Gröbner Basis and ideal membership testing.

The given circuit C is modeled as a system of polynomials $F = \{f_1, \ldots, f_s\} \subset R = \mathbb{Q}[x_1, \ldots, x_n]$. Let f be the specification polynomial for circuit C. The circuit is found to be buggy when verification is performed. Using the rectification check theorem, we find that net x_i admits single-fix rectification. We now have to compute a polynomial function $x_i = U(X_{PI})$ such that its implementation at net x_i corrects the circuit. When RTTO is imposed on the variables of the ring, every polynomial is of the form $f_j : x_j + tail(f_j)$. We have shown in the previous chapters that a circuit implements the specification of f vanishes over the variety $V(J + J_0)$, where $J = \langle f_1, \ldots, f_s \rangle$ and $J_0 = \langle x_l^2 - x_l : l = 1, \ldots, n \rangle$. Therefore $f \in \langle f_1, \ldots, f_{i-1}, f_i : x_i - U(X_{PI}), f_{i+1}, \ldots, f_s, x_l^2 - x_l \rangle$. The polynomial $U(X_{PI})$ is the hitherto unknown rectification function which rectifies the circuit at net x_i. We compute a polynomial $U(X_{PI})$ which satisfies the above ideal membership. We define our problem statement more formally in the following section.

6.1 Problem Statement

The following is our problem statement:

- Given $R = \mathbb{Q}[x_1, \ldots, x_n]$.
- Given a word-level specification polynomial f. Note that f had integral coefficients.
- Given a buggy k-bit integer multiplier circuit C. The bit-level primary inputs are $\{a_0, \ldots, a_{k-1}\}$ and $\{b_0, \ldots, b_{k-1}\}$. The bit-level primary outputs are $\{z_0, \ldots, z_{2k-1}\}$.
- Analyze the topology of the circuit and derive RTTO.
- Given a system of polynomial equations modeling the gates in the circuit C, $F = \{f_1, \ldots, f_s\} \subset R = \mathbb{Q}[x_1, \ldots, x_n]$ and the set $F_0 = \{x_1^2 - x_1, \ldots, x_n^2 - x_n\}$.
- Given a net x_i which admits single-fix rectification.

Our goal is to find a polynomial function $x_i = U(X_{PI})$ that maps $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$, such that $f \in \langle f_1, \ldots, f_i : x_i - U(X_{PI}), \ldots, f_s, x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. This polynomial function rectifies the circuit.

Example 6.1. *Consider a buggy 2-bit multiplier in Fig. 6.1.*

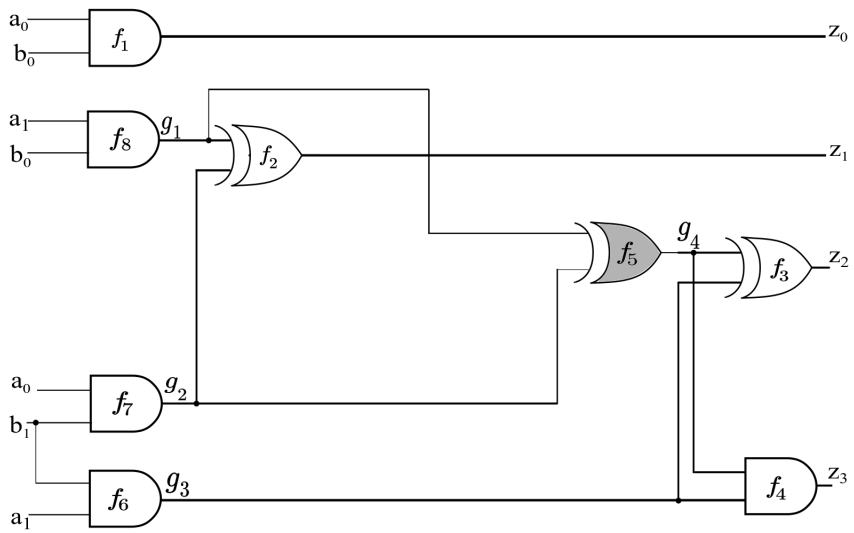

Figure 6.1: 2-bit Integer multiplier with a bug. The AND gate at g_4 replaced by an XOR gate.

- *Given the ring* $R = \mathbb{Q}[z_0, z_1, z_2, z_3, g_4, g_3, g_2, g_1, a_0, a_1, b_0, b_1]$.

- *Given the specification polynomial*
  $f = -(8 \cdot z_3 + 4 \cdot z_2 + 2 \cdot z_1 + z_0) + (2 \cdot a_1 + a_0) \cdot (2 \cdot b_1 + b_0)$

- *Given a 2-bit integer multiplier circuit C. The bit-level primary inputs are* $\{a_0, \ldots, a_1\}$ *and* $\{b_0, \ldots, b_1\}$. *The bit-level primary outputs are* $\{z_0, \ldots, z_3\}$.

- *A bug is introduced by replacing the correct AND gate at* f_6 *by an XOR gate.*

- *Analyze the topology of the circuit and derive RTTO. The term order imposed is:* $lex : \{z_0 > z_1 > z_2 > z_3\} > \{g_4\} > \{g_3 > g_2 > g_1\} > \{a_0 > a_1 > b_0 > b_1\}$

- *Given a system of polynomial equations modeling the gates in the circuit* $F = \{f_1, \ldots, f_8\}$ *and the set of polynomials* $F_0 = \{a_0^2 - a_0, \ldots, z_3^2 - z_3\}$.

- *Given net* $x_i = g_4$ *where single-fix rectification is possible.*

The polynomial function $g_4 = a_0 \cdot a_1 \cdot b_0 \cdot b_1$ *is computed such that* $f \in \langle f_1, \ldots, f_6 : g_4 - a_0 \cdot a_1 \cdot b_0 \cdot b_1, \ldots, f_8, a_0^2 - a_0, \ldots, z_3^2 - z_3 \rangle$.

In the subsequent sections, we describe how to compute a rectification polynomial that rectifies the circuit at net x_i.

6.2 Computing rectification function

Let $R = \mathbb{Q}[x_1, \ldots, x_n]$ and let $F = \{f_1, \ldots, f_i : x_i - U(X_{PI}), \ldots, f_s\}$ the system of polynomials modeling the correct circuit and $U(X_{PI})$ is the unknown rectification polynomial to be computed. The polynomials are ordered such that according to RTTO, $lt(f_1) > lt(f_2) > \cdots > lt(f_i) : x_i > lt(f_{i+1}) > \cdots > lt(f_s)$. The desired rectification function $U(X_{PI})$ computed must satisfy the condition $f \in \langle f_1, \ldots, f_i : x_i - U(X_{PI}), \ldots, f_s, x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. This polynomial function $U(X_{PI})$ can be computed using the combination of extended Gröbner Basis and ideal membership testing.

Let the polynomial $f_i = x_i - U$ correct the circuit. We can write:

$$f \in \langle f_1, \ldots, f_{i-1}, \boldsymbol{f_i : x_i - U}, f_{i+1}, \ldots, f_s, x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle \tag{6.1}$$

The ideal membership relation of f can be written as:

$$f = h_1 f_1 + h_2 f_2 + \cdots + \boldsymbol{h_i f_i} + \cdots + h_s f_s + \sum_{l=1}^{n} H_l \cdot (x_l^2 - x_l) \tag{6.2}$$

where $\{h_1, \ldots, h_s, H_l\} \subset R$ are arbitrary polynomials. Substituting $f_i = x_i - U$:

$$f = h_1 f_1 + h_2 f_2 + \cdots + \boldsymbol{h_i(x_i - U)} + \cdots + h_s f_s + \sum_{l=1}^{n} H_l \cdot (x_l^2 - x_l) \tag{6.3}$$

$$f = h_1 f_1 + h_2 f_2 + \cdots + \boldsymbol{h_i x_i - h_i U} + \cdots + h_s f_s + \sum_{l=1}^{n} H_l \cdot (x_l^2 - x_l) \tag{6.4}$$

$$f - h_1 f_1 - h_2 f_2 - \cdots - h_i x_i = -h_i U + \cdots + h_s f_s + \sum_{l=1}^{n} H_l \cdot (x_l^2 - x_l) \tag{6.5}$$

Notice that on the L.H.S. of Eqn. (6.5), the polynomials $f, f_1, \ldots, f_{i-1}$ and the monomial x_i are known polynomial expressions. Therefore, f can be divided by $f_1, \ldots, f_{i-1}$ and x_i to obtain the respective quotients of the division $h_1, \ldots, h_i$ and remainder r. The polynomial f can be written as

$$f = h_1 f_1 + h_2 f_2 + \cdots + h_i x_i + r \tag{6.6}$$

The remainder r can be computed as:

$$r = f - h_1 f_1 - h_2 f_2 - \cdots - h_i x_i \tag{6.7}$$

After h_i is computed (as quotient of division by x_i), the R.H.S. of Eqn. (6.5) consists of $h_i, f_{i+1}, \ldots, f_s$ and polynomials $x_l^2 - x_l$ as known expressions. This implies:

$$f - h_1 f_1 - \cdots - h_i x_i \in \langle -h_i, f_{i+1}, \ldots, f_s, x_l^2 - x_l \rangle \tag{6.8}$$

$$r \in \langle -h_i, f_{i+1}, \ldots, f_s, x_l^2 - x_l \rangle \tag{6.9}$$

Note that RTTO renders the polynomial $f_{i+1}, \ldots, f_s$ a Gröbner Basis. However, $\{-h_i, f_{i+1}, \ldots, f_s\}$ is not. We compute its a Gröbner Basis $G = \{g_1, \ldots, g_t\}$. We

apply the extended Gröbner Basis theory explained in Eqn. (3.16) to G to find the ideal membership relation of r.

$$r = -h_i' h_i + h_{i+1}' f_{i+1} + \cdots + h_s' f_s + \sum H_l'(x_l^2 - x_l) \tag{6.10}$$

Then $U = h_i'$ is a polynomial function that rectifies the circuit such that the circuit matches the specification. This polynomial rectifies the circuit such that the circuit matches the specification.

The rectification polynomial h_i' is computed over the field of rational numbers $\mathbb{Q}$. This polynomial may have fractional coefficients. The rectification polynomial h_i' is computed as quotient of division. The quotient q in one step division $f \xrightarrow{g} r$ is calculated as $r = f - \frac{lt(f)}{lt(g)}g$. The quotient is equal to $\frac{lt(f)}{lt(g)}$. If $lt(g)$ does not completely divide $lt(f)$, the quotient computed may contain be a fraction. Therefore, our procedure may introduce fractional coefficients in the polynomial h_i'. This polynomial may also evaluate to any value in $\mathbb{Q}$ for some primary input vectors. We use the example of a 3-bit multiplier shown in Fig. 6.2 to explain this in more detail.

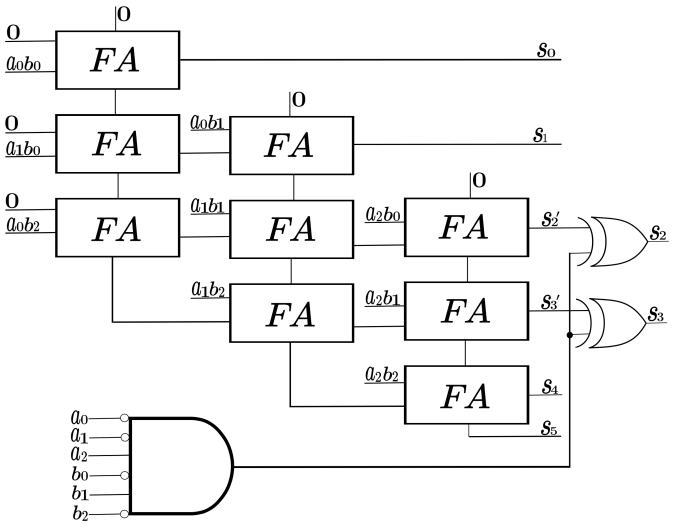

Figure 6.2: 3-bit multiplier with additional circuit to introduce a bug

Example 6.2. *The Fig. 6.2 shows a 3-bit array multiplier circuit. A bug in introduced in the circuit in the form of additional gates as shown. Gates are added to the circuit to*

modify the function of the circuit at two output bits for one input pattern. We find that single-fix rectification is feasible at a net n_{22} in the circuit. We apply Eqns. (6.1) to (6.10) and compute the rectification function. The following rectification function is computed. We also computed h_i as the quotient of division by x_i, as shown in Eqn. (6.5).

$$\begin{aligned} h_i = {} & 8 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 + 16 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 - 12 \cdot a_0 \cdot a_1 \\ & - 8 \cdot a_0 \cdot a_2 \cdot b_0 - 16 \cdot a_0 \cdot a_2 \cdot b_1 + 12 \cdot a_0 - 8 \cdot a_1 \cdot a_2 \cdot b_0 \\ & - 16 \cdot a_1 \cdot a_2 \cdot b_1 + 12 \cdot a_1 + 8 \cdot a_2 \cdot b_0 + 16 \cdot a_2 \cdot b_1 - 12 \end{aligned}$$

$$\begin{aligned} h_i' = {} & -\frac{44}{3} \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + \frac{4}{3} \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \\ & + 8 \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 + \frac{8}{3} \cdot a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 \\ & + \frac{4}{3} \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 \cdot b_2 - \frac{2}{3} \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 \\ & + \frac{4}{3} \cdot a_0 \cdot a_1 \cdot b_1 \cdot b_2 + \frac{8}{3} \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\ & - 4 \cdot a_0 \cdot a_2 \cdot b_0 \cdot b_2 + \frac{28}{3} \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 \\ & - \frac{14}{3} \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 - \frac{8}{3} \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 \\ & - \frac{20}{3} \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 + \frac{8}{3} \cdot a_1 \cdot a_2 \cdot b_1 \\ & - \frac{2}{3} \cdot a_1 \cdot b_1 - \frac{4}{3} \cdot a_1 \cdot b_2 + 1 \end{aligned}$$

We evaluate the polynomial h_i and h_i' for all primary input patterns. The observations are recorded in the following table.

$a_0, a_1, a_2, b_0, b_1, b_2$	h_i	h_i'
0,0,0,0,0,0	-12	1
0,0,0,0,0,1	-12	1
0,0,0,0,1,0	-12	1
0,1,0,0,0,1	0	$-\frac{1}{3}$
0,1,0,0,1,0	0	$\frac{1}{3}$
0,1,0,0,1,1	0	-1

Table 6.1: Evaluating h_i and h_i'

In order to conserve space, Table 6.1 shows the data for only some input patterns. We make some observations based on the contents of this table. The rectification polynomial h_i' has fractional coefficients. At points where $h_i = 0$, we see that h_i' can evaluate to any value in $\mathbb{Q}$. We found 48 such points where h_i evaluates to 0. At points where $h_i \neq 0$, we observe that h_i' evaluates to a value in $\{0, 1\}$.

Consider the polynomial h_i which is computed as a quotient of division by x_i in Eqn. (6.5). We can see that the rectification polynomial h_i' depends on the polynomial h_i. This is also confirmed by the data shown in Table 6.1. In order to understand the conditions under which the rectification polynomial h_i' implements a Boolean function, we look at the variety of h_i.

Let us assume there exists a polynomial U, which can be implemented at net x_i to rectify the circuit. The existence of such a polynomial is proved by application of Theorem 5.1. This polynomial implements a Boolean function, which performs the mapping $\{0,1\}^{|X_j|} \rightarrow \{0,1\}$, where X_j is the set of variable that are less than x_i in RTTO. This polynomial can be reduced to the primary input variables, $U(X_{PI})$. Based on these assumptions, we state the following facts.

Fact: The polynomial h_i is computed in terms of variables in the set X_j, where all elements in $X_j < x_i$ in RTTO.

We use Gröbner basis division to compute h_i. This division is shown in Eqn. (6.5). The order of division ensures that h_i is a polynomial in those variables which are less than x_i in the term order.

Fact: The polynomial h_i' is computed in terms of variables in the set X_j, where all elements in $X_j < x_i$ in RTTO.

During division of f by polynomials $f_1, \ldots, f_{i-1}, x_i$, shown in Eqn. (6.5), the order of division follows RTTO. The leading terms of polynomials $\{f_1, \ldots, f_{i-1}\}$ correspond to the nets in the circuit that appear earlier in the topological level than x_i, such that $lt(f_1) > \cdots > lt(f_{i-1}) > lt(f_i) : x_i$. Algorithmically, as $U = h_i'$ is computed as a quotient of division, U may contain any variables $X \subseteq \{x_1, \ldots, x_n\}$ in its support. However, due to the imposition of our term order $>$, U will contain only those variables $X_j \subset \{x_1, \ldots, x_n\}$ in its support set that are less than x_i in the

term order.

Fact:

$$h_i(U - h_i') = \sum_{j=i+1}^{s} (h_j - h_j')f_j + \sum_{x_l \in X_{PI}} (H_l - H_l')(x_l^2 - x_l) \tag{6.11}$$

We get Eqn. (6.11) by combining Eqns. (6.5), (6.7) and (6.10).

Fact: Let $p \in \{0,1\}^{|X_{PI}|}$ be an assignment to the primary inputs of C. Let p' be the assignment to the all the nets in C under the application of p to X_{PI}; denote $p'|_{X_{PI}} = p$ as the projection of p' to primary inputs. Then, we have $\forall p \in \{0,1\}^{|X_{PI}|}$, $\exists p'$, such that $p'|_{X_{PI}} = p$ and $f_{i+1}(p') = \cdots = f_s(p') = 0$.

Clearly, for all primary input assignments, there exists an assignment to the nets that satisfies all the gates of the circuit. By using the three facts stated above, we state and prove the following results:

Theorem 6.1. *Let there exist a polynomial U that maps from $\{0,1\}^{|X_j|} \rightarrow \{0,1\}$. Let p' be a point such that $p'|_{X_{PI}} = p$ and $f_{i+1}(p') = \cdots = f_s(p') = 0$. If $h_i(p') \neq 0$, then $h_i'(p) = U(p')$.*

Proof: Consider Eqn. (6.11). Consider a point p' such that $p'|_{X_{PI}}$ and $f_{i+1}(p') = \cdots = f_s(p') = 0$. We know that $x_l^2 = x_l$ at any point $p \in \{0,1\}^{|X_{PI}|}$. Therefore, R.H.S. of Eqn. (6.11) is 0.

$$h_i(U - h_i') = 0. \tag{6.12}$$

In L.H.S. of Eqn. (6.12), since $h_i(p') \neq 0$, $U(p') = h_i'(p')$.

Since we know that U is a Boolean polynomial, we know that $U(p')$ is a value in $\{0,1\}$. Therefore, according to Theorem 6.1, h_i' evaluates to a Boolean value at those points where $h_i \neq 0$.

Corollary 6.1. *Let point p be such that $p \in \{0,1\}^{|X_{PI}|}$. Let p' be a point such that $p'|_{X_{PI}} = p$. If $h_i(p') = 0$, the polynomial h_i' can be modified such that $h_i'(p')$ evaluates to a value in $\{0,1\}$.*

Proof: Consider Eqn. (6.11). Consider a point p' such that $p'|_{X_{PI}}$ and $f_{i+1}(p') = \cdots = f_s(p') = 0$. We know that $x_l^2 = x_l$ at any point $p \in \{0,1\}^{|X_{PI}|}$. Therefore,

R.H.S. of Eqn. (6.11) is 0.

$$h_i(U - h_i') = 0. \tag{6.13}$$

In L.H.S. of Eqn. (6.13), $h_i(p') = 0$. Therefore, it does not depend on $h_i'(p')$.

The corollary for Theorem 6.1 implies that we can modify the polynomial function h_i' at those points p, where $h_i(p) = 0$, to evaluate to any value in $\{0, 1\}$, without affecting the ideal membership relation in Eqn (6.9). These points are the local don't cares of the rectification function.

Due to the compact structure of arithmetic circuits, our computation usually results in a h_i such that $V(\langle h_i \rangle + J_0) = \emptyset$. This implies that the polynomial h_i does not evaluate to 0 at any point in $\{0, 1\}^{|X_{PI}|}$.

Corollary 6.2. *If $\forall p \in \{0,1\}^{|X_{PI}|}$ and $h_i(p') \neq 0$ $\forall p'$ such that $p'|_{X_{PI}} = p$, then polynomial U is unique and can be computed as polynomial h_i' by application of Eqns. (6.1) to (6.10).*

Proof: By extension of proof for Theorem 6.1, $U(p') = h_i'(p')$, $\forall p'$.

If the application of our procedure computes a Boolean rectification polynomial, then we show a procedure to synthesize the polynomial as a circuit.

Proposition 6.1. *The rectification polynomial h_i' can be normalized to a polynomial in primary input variables by computing $U(X_{PI}) = h_i' \pmod{J + J_0}$ computed as $h_i' \xrightarrow{J+J_0} U(X_{PI})$. This polynomial will have integral coefficients and maps from $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$.*

The unique rectification function that is computed by our approach implements a Boolean function. A polynomial with that property will only have integral coefficients and not fractional coefficients.

When $h_i' \pmod{J + J_0}$ is computed, the internal variables are reduced to primary inputs. Because the primary input variables are last in RTTO, they do not get cancelled. Therefore, $U(X_{PI})$ is a polynomial in primary input variables, where every monomial is a multi-linear expression. For all values in $\{0,1\}^{|X_{PI}|}$, $U(X_{PI})$ evaluates to a value in $\{0, 1\}$.

In this fashion, the polynomial $f_i : x_i - U(X_{PI})$ can be identified as a rectifi-

cation polynomial for the circuit. The rectification polynomial we compute is a polynomial in $\mathbb{Q}$ with integer coefficients and is not easy to synthesize as a circuit. Therefore, we compute $U(X_{PI}) \pmod 2$ and treat addition (+) as XOR operation and multiplication ($\cdot$) as AND operation. This AND-XOR expression is given a synthesis tool to generate a logic circuit. The implementation of this circuit at net x_i rectifies the circuit. The rectified circuit becomes functionally equivalent to specification f.

6.2.1 Procedure to compute rectification function

Eqn. (6.1) to Eqn. (6.5) can be applied using the steps enumerated below.

- Analyze circuit topology and obtain a term order $>$.
- Order the polynomials of the equation according to the term order $F = \{f_1, \ldots, f_s\}$, such that $lt(f_1) > lt(f_2) > \cdots > lt(f_i) : x_i > \cdots > lt(f_s)$.
- Apply Eqn. (6.5) by performing the division $f \xrightarrow{f_1,\ldots,f_{i-1},x_i} r$. Record the quotient of division by x_i as h_i.
- Apply Eqn. 3.16 to find r in terms of $\{h_i, f_{i+1}, \ldots, f_s\}$ and compute the polynomial h_i' as shown in Eqn. (6.10).
- Let ideal $J_q = \langle h_i \rangle + J_0$. Compute Gröbner Basis $G = GB(J_q)$.
- **Case 1:** $G = \{1\}$,
  - Reduce h_i' modulo $(F \cup F_0)$ to obtain the solution in terms of primary inputs, $U(X_{PI})$.
  - Compute $U(X_{PI}) \pmod 2$. Interpret addition (+) as XOR and multiplication ($\cdot$) as AND. Use a synthesis tool to generate a logic circuit from this AND-XOR expression.
- **Case 2:** $G \neq \{1\}$
  - Evaluate h_i for all points p'

- Where $h_i(p') = 0$, set $h_i'(p) = U(p)$ as *don't cares*
- Generate a function table
- Synthesize and implement function at net x_i

This procedure can be explained using an example of the same buggy 3-bit multiplier implementation from Fig. 4.1.

Example 6.3. *Consider the 3-bit integer multiplier shown in Fig. 4.1. The variables* $a_0, a_1, a_2, b_0, b_1, b_2$ *are primary inputs,* $s_0, s_1, s_2, s_3, s_4, s_5$ *are primary outputs. The specification polynomial for this circuit is:*

$$f = -(1 \cdot s_0 + 2 \cdot s_1 + 4 \cdot s_2 + 8 \cdot s_3 + 16 \cdot s_4 + 32 \cdot s_5) + (1 \cdot a_0 + 2 \cdot a_1 + 4 \cdot a_2) \cdot (1 \cdot b_0 + 2 \cdot b_1 + 4 \cdot b_2)$$

Circuit topology is analyzed and a term order is derived. The term order imposed on the variables the circuit is:

$s_0 > s_1 > s_2 > s_3 > s_4 > s_5 > n_{40} > n_{35} > n_{38} > n_{36} > n_{30} > n_{31} > n_{17} > n_{22} > n_{37} > n_{24} > n_{29} > n_{41} > n_{33} > n_{34} > n_{25} > n_{28} > n_{26} > n_{20} > n_{21} > n_{14} > n_{15} > n_{27} > n_{18} > n_{19} > a_0 > a_1 > a_2 > b_0 > b_1 > b_2$

Let a bug be introduced at net n_{38} *by replacing the correct OR gate with an AND gate. The polynomials of the circuit are ordered according to the term order imposed and are repeated here:*

$$f_0 = m_0 - a_0 \cdot b_0$$
$$f_1 = m_1 + 2 \cdot n_{14} \cdot n_{15} - n_{14} - n_{15}$$
$$f_2 = m_2 + 2 \cdot n_{17} \cdot n_{22} - n_{17} - n_{22}$$
$$f_3 = m_3 + 2 \cdot n_{30} \cdot n_{31} - n_{30} - n_{31}$$
$$f_4 = m_4 + 2 \cdot n_{35} \cdot n_{38} - n_{35} - n_{38}$$
$$f_5 = m_5 + n_{40} \cdot n_{41} - n_{40} - n_{41}$$
$$f_6 = n_{40} - n_{35} \cdot n_{38}$$

$$f_7 = n_{35} + 2 \cdot n_{33} \cdot n_{34} - n_{33} - n_{34}$$

$$\boldsymbol{f_8 = n_{38} - n_{36} \cdot n_{37}}$$

$$f_9 = n_{36} - n_{30} \cdot n_{31}$$

$$f_{10} = n_{30} + 2 \cdot n_{24} \cdot n_{29} - n_{24} - n_{29}$$

$$f_{11} = n_{31} - n_{17} \cdot n_{22}$$

$$f_{12} = n_{17} - a_0 \cdot b_2$$

$$f_{13} = n_{22} + 2 \cdot n_{20} \cdot n_{21} - n_{20} - n_{21}$$

$$f_{14} = n_{37} - n_{24} \cdot n_{29}$$

$$f_{15} = n_{24} - a_1 \cdot b_2$$

$$f_{16} = n_{29} + 2 \cdot n_{25} \cdot n_{28} - n_{25} - n_{28}$$

$$f_{17} = n_{41} - n_{33} \cdot n_{34}$$

$$f_{18} = n_{33} - a_2 \cdot b_2$$

$$f_{19} = n_{34} - n_{25} \cdot n_{28}$$

$$f_{20} = n_{25} - a_2 \cdot b_1$$

$$f_{21} = n_{28} + n_{26} \cdot n_{27} - n_{26} - n_{27}$$

$$f_{22} = n_{26} - n_{20} \cdot n_{21}$$

$$f_{23} = n_{20} + 2 \cdot n_{18} \cdot n_{19} - n_{18} - n_{19}$$

$$f_{24} = n_{21} - n_{14} \cdot n_{15}$$

$$f_{25} = n_{14} - a_0 \cdot b_1$$

$$f_{26} = n_{15} - a_1 \cdot b_0$$

$$f_{27} = n_{27} - n_{18} \cdot n_{19}$$

$$f_{28} = n_{18} - a_1 \cdot b_1$$

$$f_{29} = n_{19} - a_2 \cdot b_0$$

In Example 5.4, we found that net n_{38} admits single-fix rectification. Now let us find a rectification polynomial U that can correct the bug in the circuit.

Reduction according to Eqn. (6.5) is done as follows:

$f \xrightarrow{f_1} \xrightarrow{f_2} \xrightarrow{f_3} \xrightarrow{f_4} \xrightarrow{f_5} \xrightarrow{f_6} \xrightarrow{f_7} \xrightarrow{n_{38}} r$

$r = 16 \cdot n_{30} \cdot n_{31} - 8 \cdot n_{30} - 8 \cdot n_{31} + 8 \cdot n_{17} \cdot n_{22} - 4 \cdot n_{17} - 4 \cdot n_{22} - 32 \cdot n_{41} + 32 \cdot$

$n_{33} \cdot n_{34} - 16 \cdot n_{33} - 16 \cdot n_{34} + 4 \cdot n_{14} \cdot n_{15} - 2 \cdot n_{14} - 2 \cdot n_{15} + 2 \cdot a_0 \cdot b_1 + 4 \cdot a_0 \cdot b_2 + 2 \cdot a_1 \cdot b_0 + 4 \cdot a_1 \cdot b_1 + 8 \cdot a_1 \cdot b_2 + 4 \cdot a_2 \cdot b_0 + 8 \cdot a_2 \cdot b_1 + 16 \cdot a_2 \cdot b_2$

The quotient of division by n_{38} is computed as:

$h_i = -64 \cdot n_{41} \cdot n_{33} \cdot n_{34} + 32 \cdot n_{41} \cdot n_{33} + 32 \cdot n_{41} \cdot n_{34} - 16$

Therefore,

$r \in \langle h_i, f_9, \ldots, f_{29} \rangle$

Applying the extended Gröbner Basis theory from Eqn. (3.16), we compute the following polynomial as rectification polynomial U:

$$\begin{aligned} h_i' = {} & n_{30} \cdot n_{31} - \frac{1}{2} \cdot n_{30} - \frac{1}{2} \cdot n_{31} + \frac{1}{2} \cdot n_{17} \cdot n_{22} - \frac{1}{4} \cdot n_{17} - \frac{1}{4} \cdot n_{22} - \\ & 2 \cdot n_{41} + 2 \cdot n_{33} \cdot n_{34} - n_{33} - n_{34} + \frac{1}{4} \cdot n_{14} \cdot n_{15} - \frac{1}{8} \cdot n_{14} - \\ & \frac{1}{8} \cdot n_{15} + \frac{1}{8} \cdot a_0 \cdot b_1 + \frac{1}{4} \cdot a_0 \cdot b_2 + \frac{1}{8} \cdot a_1 \cdot b_0 + \frac{1}{4} \cdot a_1 \cdot b_1 + \frac{1}{2} \cdot a_1 \cdot b_2 + \\ & \frac{1}{4} \cdot a_2 \cdot b_0 + \frac{1}{2} \cdot a_2 \cdot b_1 + a_2 \cdot b_2 \end{aligned} \tag{6.14}$$

Consider ideal $J_q = \langle h_i \rangle + J_0$. We compute $G = GB(J_q) = \{1\}$. This indicates that the polynomial $h_i' = U$ and it is a unique polynomial that maps from $\{0,1\}^{|X_j|} \rightarrow \{0,1\}$

As we can observe, the polynomial U comprises monomials in both primary inputs as well as internal net variables. It also has fractional coefficients. Reducing this polynomial $(\text{mod } J + J_0)$ *gives the polynomial below.*

$U(X_{PI}) = -a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 - a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 + a_0 \cdot a_1 \cdot b_1 \cdot b_2 + a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 - a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + a_1 \cdot a_2 \cdot b_1 \cdot b_2$

This polynomial $U(X_{PI})$ is only in primary inputs.

We can verify that $f \in \langle f_1, \ldots, f_8 : n_{38} - U(X_{PI}), \ldots, f_{29} \rangle$. This indicates that the polynomial computed rectifies the circuit.

Compute $U(X_{PI})$ $(\text{mod } 2)$.

$U(X_{PI})$ $(\text{mod } 2) = a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + a_0 \cdot a_1 \cdot a_2 \cdot b_0 \cdot b_2 + a_0 \cdot a_1 \cdot a_2 \cdot b_1 \cdot b_2 + a_0 \cdot a_1 \cdot b_1 \cdot b_2 + a_0 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + a_1 \cdot a_2 \cdot b_0 \cdot b_1 \cdot b_2 + a_1 \cdot a_2 \cdot b_1 \cdot b_2$

In this polynomial, addition (+) is interpreted as XOR and multiplication (·) as AND/. This AND-XOR expression is given to a synthesis tool and a logic circuit is generated.

In the above example, this procedure was applied to a multiplier without using

the columns described in [13]. However, column-wise reduction described in this paper is also done topologically. Therefore, our procedure is applied on multipliers using the column-wise incremental reduction without altering the basic principles of ideal membership testing and extended Gröbner Basis theory used in the procedure.

6.3 Experimental setup and results

We have performed experiments on integer array multiplier circuits where the implementation is different from the specification due to the presence of a bug. We generated AIG files of array multipliers up to 64 bits by using the *gen -m* command from ABC [10]. The tool from the authors in [13] was used to input the AIG files and obtain *.SING files for the multiplier. We introduced bugs which could be corrected by a single-fix rectification. The complete single-fix rectification procedure is implemented using the SINGULAR symbolic algebra computation system [ver. 4-1-0][[56]]. The experiments were conducted on a desktop computer with a 3.5GHz Intel CoreTM i7-4770K Quad-core CPU, 16 GB RAM, running 64-bit Linux OS.

k	t_1	t_2	t_3	t_4
2	0.002	0.003	0.007	0.005
4	0.005	5.820	0.024	5.824
8	0.027	9.103	0.206	9.111
12	0.120	23.137	0.641	23.158
16	0.400	42.915	1.782	42.981
18	0.647	48.964	2.479	49.060
28	6.329	288.448	26.707	289.860
32	12.119	368.319	44.965	370.579
56	292.203	980.283	1221.654	1040.504
64	577.162	16.049	28.597	20.147

Table 6.2: Single-fix rectification of integer multiplier against polynomial specification. Time in seconds; k = Datapath size, t_1 = verification time, t_2 = time to find potentially rectifiable nets , t_3 = time for rectification check, t_4 = time to compute rectification function. Time Out = 10800 seconds.

In the above table, t_1 is the time taken to perform verification of the buggy

circuits using the reduction model described in [13]. Time t_2 indicates the application of Proposition. 5.1 to find set of potentially rectifiable nets, $\mathcal{N} \subset \{x_1, \ldots, x_n\}$. Time t_3 indicates the time taken to apply Theorem. 5 on the nets in the set $\mathcal{N}$ till a rectification target net $x_i \subset \mathcal{N}$ is found. And time t_4 is the time taken to compute a rectification polynomial using our approach.

An interesting observation in Table 6.2 is the difference in computation time between the 56-bit multiplier and 64-bit multiplier. Single-fix rectification in the 56-bit multiplier is attempted at a net x_i that is located deeper in the circuit, when compared to the placement of the rectification target in the 64-bit multiplier. This result shows that the efficiency of our approach depends on the location of the rectification target net.

Once our approach computes a rectification polynomial, we also describe how we can obtain an AND-XOR expression of the rectification function. When this function is implemented in the buggy circuit, it is desirable to compare the area and delay properties of the original circuit and the rectified circuit. We performed some experiments to obtain synthesis results for rectified circuits, compared to the original correct circuit.

The AIG files which were used to generate the SINGULAR files are given as input to ABC. The BLIF files are generated for these multipliers. The circuit is then mapped to a logic library, *"cadence.genlib1"*, consisting of INVERTER, BUFFER, AND, OR and XOR gates. The command *"print_stats"* was used to obtain area and delay properties of the circuits.

A bug b_1 was introduced in the circuit and single-fix rectification was attempted. We computed a rectification polynomial and obtained an AND-XOR expression for the rectification function. This rectification function was written in BLIF format and used to rectify the bug in the BLIF file. We then use the command "*resyn*" in ABC to re-synthesize the circuit. This circuit was then mapped to the same logic library with INVERTER, BUFFER, AND, OR and XOR gates. The command *"print_stats"* was used to obtain area and delay properties of the rectified circuit. To find the dependency of synthesis results on location of the bug, the above experiment was repeated for other bugs b_2 and b_3. The bug b_2 is located deeper

in the circuit compared to b_1, and bug b_3 is located deeper in the circuit compared to b_2. The synthesis results are presented in Table 6.3.

k	16			32			56		
	b_1	b_2	b_3	b_1	b_2	b_3	b_1	b_2	b_3
$area_{(func)}$	3	9	1100	9	41	312	9	87	TO
$delay_{(func)}$	2.00	4.00	22.00	4.00	8.00	17.00	4.00	11.00	TO
$area_{(rect)}$	1537	1486	2492	6032	6186	6469	18608	18938	TO
$delay_{(rect)}$	88.00	89.00	89.00	184.00	184.00	184.00	328.00	328.00	TO
$penalty_{(area)}$	9.16%	5.53%	76.98%	2.44%	5.06	9.86%	1.30%	3.1%	TO
$area_{(orig)}$	1408.00			5888.00			18368.00		
$delay_{(orig)}$	88.00			184.00			328.00		

Table 6.3: Synthesis Results. *area* in sq. units. $delay$ = No. of gates.

$area_{(func)}$ and $delay_{(func)}$ are the area and delay numbers for the rectification function. $area_{(rect)}$ and $delay_{(rect)}$ are the area and delay numbers for the rectified circuit. $area_{(orig)}$ and $delay_{(orig)}$ are area and delay numbers for the original correct circuit. Note that in all cases, the rectified circuit has a larger area compared to the original correct circuit. For each bug, we have calculated the area penalty we pay for rectification by comparing the rectified circuit to the original correct circuit. For example, the rectification of a bug b_1 in the 16-bit multiplier results in a circuit with 9.1% more area compared to the original circuit. Arithmetic circuits are custom designed. However, we are synthesizing a sub-function to implement in the multiplier circuit and synthesizing the whole circuit. Logic synthesis for arithmetic circuits is not very efficient and that is reflected in these results.

Consider the bug b_3 in the 56-bit array multiplier. The bug is placed in the middle of the circuit and the Gröbner Basis reduction for verification times out. Therefore, a feasible rectification polynomial was not computed to rectify this bug. By observing the results shown in Table 6.2 and Table 6.3, we can conclude that the location of the bug, and consequently the location of the rectification target, influences the efficiency of our approach. To understand this dependency, we have conducted more experiments on a 16-bit integer multiplier by rectifying various bugs present at different locations in the circuit.

Recall that we use the column-wise incremental reduction model from [13] to perform Gröbner Basis reduction in our procedure. This model first divides a k-bit integer multiplier into $2k$ slices column-wise. The slices for a 3-bit array multiplier is shown in Fig. 4.1.

Following this model, a 16-bit integer can be divided into 32 slices, C_0 to C_{31}. We introduced a bug in slice C_2 at a net located close to the primary output s_2. Net x_i is selected so it is also located close to the primary output and admits single-fix rectification. We applied the rectification check, described in Theorem. 5.1, on net x_i. We recorded the time taken to complete the Gröbner Basis reductions required to perform the check. We repeated this procedure for bugs located in the middle of the column structure, as well as for bugs located close to the primary inputs. We repeated these experiments for bugs introduced in other columns, moving towards C_{31}. The results of the experiments are shown in Table 6.4.

	NO	NM	NI
C_2	2.310	2.188	2.289
C_4	3.197	2.673	2.286
C_6	202.160	26.586	2.087
C_{10}	TO	TO	2.162
C_{12}	TO	TO	2.252

Table 6.4: Time recorded in seconds. Time Out = 10800 seconds. NO = Near Output. NM = Near Middle. NI = Near Input.

The numbers recorded in Table 6.4 are the computation time for for applying Theorem 5.1 on net x_i. The computation times for bugs placed near output (NO), near middle (NM), and near input (NI) are compared. We observe that the Gröbner Basis reductions for rectification check does not complete for bugs placed near the primary outputs in slice C_{10}. However, reduction does complete for circuits where the bug is placed close to the primary inputs. The results in Table 6.4 indicates that the efficiency of our approach is dependent on the placement of the rectification target net. In cases where the bug is placed close to the primary output, we see the worst case of the complexity of the Gröbner Basis reduction, which is discussed further in the subsequent section.

6.4 Limitation of our approach

This thesis has so far presented an efficient framework for single-fix rectification of integer multipliers using concepts from symbolic algebra. Experiments have demonstrated the effectiveness of the algorithms in cases where single-fix rectification is feasible. The techniques, however, suffer from a few limitations.

6.4.1 Time and space explosion

Our problem suffers from the exponential worst case complexity of Gröbner basis reduction in buggy circuits. This explosion can be explained by the following example.

Example 6.4. *Consider a chain of OR gates as shown in Fig. 6.3.*

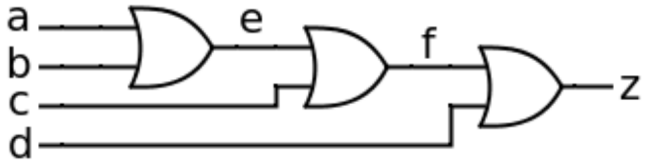

Figure 6.3: Chain of three OR gates

The polynomials for the gates in the above circuits are:

$$f_1 : z + fd - f - d$$
$$f_2 : f + ec - e - c$$
$$f_3 : e + ba - b - a$$

Let us perform the reduction $z \xrightarrow{f_1} \xrightarrow{f_2} \xrightarrow{f_3} r$.

$$\begin{aligned} z &\xrightarrow{f_1} -fd + f + d \\ &\xrightarrow{f_2} ecd - ec - ed + e - cd + c + d \\ &\xrightarrow{f_3} -abcd + abc + abd - ab + acd - ac - ad + a + bcd - bc - bd + b - cd + c + d \end{aligned}$$

The remainder computed in Ex. 6.4 contains 15 terms. This shows that if we have a chain of OR gates in the circuit, we see the worst case of Gröbner Basis reduction complexity. Polynomial representations for OR-dominated functions include more monomial terms and also more occurrences of variables among the terms. This eventually results in size-explosion of the intermediate (remainder) polynomials in the reduction. Some data on this explosion can be observed in our approach in a 8-bit integer multiplier as shown in the example below.

Example 6.5. *Consider an 8-bit integer array multiplier. It has bit-level primary inputs $a_0, \ldots, a_7$ and $b_0, \ldots, b_7$. The primary outputs of this circuit are $z_0, \ldots, z_{15}$. An 8-bit multiplier can be divided into 16 column-wise slices, whose polynomials are denoted as $G_0, \ldots, G_{15}$. Let us introduce a bug in column G_7 close to the primary output z_7. We apply the rectification check from Theorem. 5.1 to net x_i to check whether it admits single-fix or not. The net x_i lies in column G_7. Let us construct two ideals G_{7_L} and G_{7_H}, as shown in Eqn. (5.1). We perform reduction as shown in Eqn. (5.2).*

Consider the partial reduction $f \xrightarrow{G_{15},\ldots,G_{7_L}} C_7$.

The remainder C_7 has 16379 terms. When further reduction $C_7 \xrightarrow{G_6} C_6$ is attempted, we observe that the size-explosion occurs and this step does not terminate.

We have observed that the success of rectification relies on the location of the bug. More generally, our approach cannot rectify bugs located close to the output in the dense part of the implementation, i.e, at close to output z_i where i is approximately $k/2$.

This limitation is also observable in verification of buggy integer circuits. This is a result of the limitation of Gröbner Basis reduction, as discussed in this section.

6.4.2 Exploring multiple rectification functions

In Eqn. 6.10, we observe that $U = h_i'$ gives a rectification polynomial that rectifies the circuit. However, due to the presence of don't-cares, h_i' may not be a unique solution. There may exist multiplier polynomials that satisfies the ideal membership test. Our approach replies on the extended Euclidean algorithm described in [26] to compute the rectification polynomial, and it has its own heuristics

implemented in the computer tool being used. We have no control over the h_i computed. Different rectification polynomials may lead to different area synthesis trade-offs. Our algorithm does not have the leverage to explore multiplier rectification functions.

6.4.3 Rectification polynomial in terms of internal variables

Given a circuit with a black box and the immediate inputs X_{int} to the box, our approach cannot determine if there exists a rectification polynomial in terms of X_{int}.

6.5 Conclusion

Given a net x_i which admits single-fix rectification, this chapter described a procedure to compute a rectification polynomial which rectifies the circuit such that the corrected circuit exactly implements the given specification. We described the conditions under which this rectification polynomial maps from $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$. We proved that a rectification polynomial with that property has a unique representation with integer coefficients that can be used to synthesize a sub-circuit that can be implemented in the given implementation at net x_i. We demonstrated the efficacy of our procedure by conducting experiments on integer multiplier circuits up to data path size of 64 bits. We also described the limitation of our approach in rectifying certain bugs based on the placement of the bug in the circuit.
i

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This thesis has presented a comprehensive framework for single-fix rectification of integer arithmetic circuits. This thesis presents novel techniques based on computer-algebra and algebraic-geometry to compute a rectification function at a particular net to rectify the bug identified from verification. We target two specific problems: (i) Identify a subset of nets where single-fix rectification may be possible and then subsequently select a net which we ascertain admits single-fix rectification; and (ii) compute a rectification polynomial that can be implemented as sub-circuit in the implementation to correct the bug. Our approach is formulated using Gröbner basis theory as a decision procedure and as a quantification procedure.

7.1 Rectification of integer arithmetic circuits

The arithmetic circuits are modeled as a polynomial system in the ring $R = \mathbb{Q}[x_1, \ldots, x_n]$. The gates in the circuit as a system of multivariate polynomials $F = \{f_1, \ldots, f_s\}$. The specification is given as a polynomial f in $\mathbb{Q}$. The topology of the circuit is analyzed and a Reverse Topological Term Order is derived. To process the polynomials systematically, RTTO is imposed on the variables of the ring. RTTO renders the polynomials itself a Gröbner basis. Verification is formulated as Gröbner basis reduction via polynomial division.

Verification of a buggy circuit results in the computation of a non-zero remainder r. By analyzing the coefficients of the remainder r, we discern the output bits which are impacted by the bug(s). Subsequently, we narrow the search space for rectifiable nets from the entire circuit to a subset of internal nets, $\mathcal{N} \subset \{x_1, \ldots, x_n\}$, where single-fix rectification may be possible. We described a rectification check formulated using Nullstellensatz principles to ascertain the necessary and suffi-

cient conditions for a net to admit single-fix rectification. We apply the rectification check to select a net $x_i \subset \mathcal{N}$ where single-fix rectification is determined to be possible. The rectification check confirms the existence of polynomial function that maps from $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$ which can be implemented at x_i to rectify the bug (bugs).

This thesis describes a procedure to compute a rectification polynomial to rectify the circuit. The procedure is formulated using ideal membership testing and extended Gröbner Basis theory. This rectification polynomial has a Boolean function, mapping from $\{0,1\}^{|X_{PI}|} \rightarrow \{0,1\}$. We prove that such a polynomial has a unique representation with integer coefficients. We reduce this polynomial $(\text{mod } 2)$ and interpret addition (+) as XOR and multiplication ($\cdot$). This AND-XOR expression is synthesized using a synthesis tool and implemented as a sub-circuit at net x_i. This results in the rectified circuit and specification becoming functionally equivalent. We demonstrate the efficacy of our approach by conducting experiments on integer multipliers up to 64 bits.

7.2 Future Work

The approaches and theories presented in this thesis can be further extended to enhance the efficiency and usability of our single-fix rectification approach. These future research directions are proposed in this section.

7.2.1 Computing multiple rectification functions

The procedure described in this thesis computes a rectification polynomial U that satisfies the ideal membership relation for the specification polynomial, $f \in \langle f_1, \ldots, f_i : x_i - U, \ldots, f_s \rangle$. This polynomial can be synthesized as a logic circuit implemented at net x_i to rectify the circuit. Due the presence of don't cares in the circuit, there may exist more than one rectification function that can be implemented at net x_i. Consider the example of a 2-bit integer multiplier below.

Example 7.1. *Consider a 2-bit integer multiplier with redundancy shown in Fig. 7.1. A bug is introduced at net I. The correct implementation has an AND gate, which is replaced by an XOR gate to introduce the bug. We confirm that single-fix rectification is possible at*

net I.

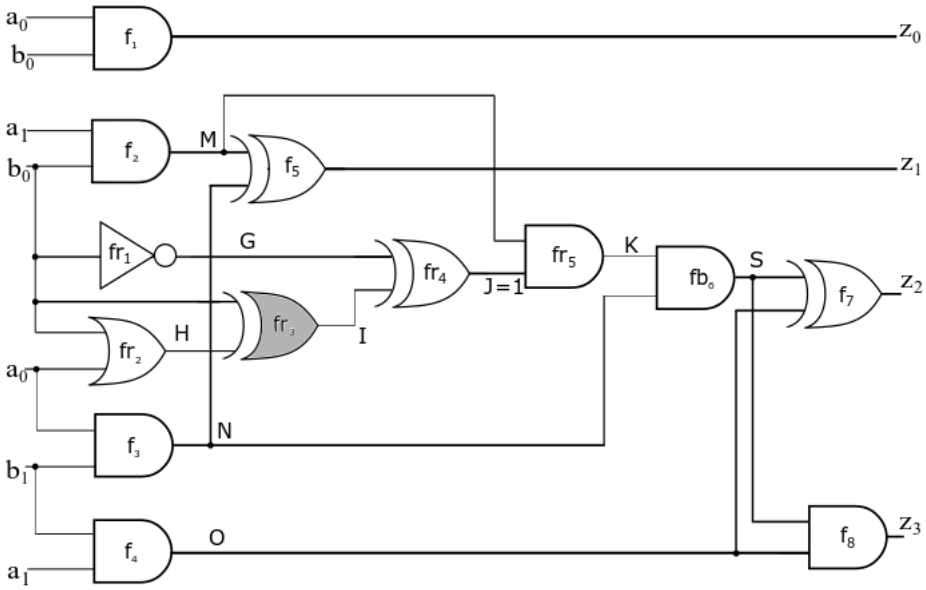

Figure 7.1: 2-bit integer multiplier with redundancy. Bug at I.

The procedure to compute a rectification polynomial is applied and the polynomial $I = b_0$ is computed.

For the above example, our procedure computed a rectification polynomial $U = b_0$. There exists a polynomial different $U' = 1$ that rectifies the circuit if implemented at net I. There exists a polynomial $U'' = a_0 \cdot a_1 \cdot b_0 \cdot b_1$ that also rectifies the circuit. The choice of the rectification function can have important synthesis consequences. Therefore, the algorithmic computation of multiple single-fix rectification polynomials is a relevant research problem which requires further study.

7.2.2 Dependency constrained rectification function

Our procedure computes a rectification polynomial in terms of primary input variables. It is desirable in partial synthesis of rectification function to find a function in terms of the internal nets to facilitate logic reuse in the circuit. Let X_{int} be a subset of the internal nets in the circuit, which the rectification polynomial U we compute must be dependent on. If polynomial U can be written in terms of

X_{int}, we can find it by changing the term order with X_{int} last in order. However, if polynomial U cannot be written in terms of X_{int}, it is desirable to find a polynomial U', which can be written in terms of X_{int}.

Example 7.2. *Consider the 2-bit integer multiplier in Fig. 7.2, with a bug at net s. A bug is introduced at net s by replacing the AND gate by an OR gate. Suppose we need to find a polynomial U that can be written in terms of* $X_{int} = \{k, n\}$.

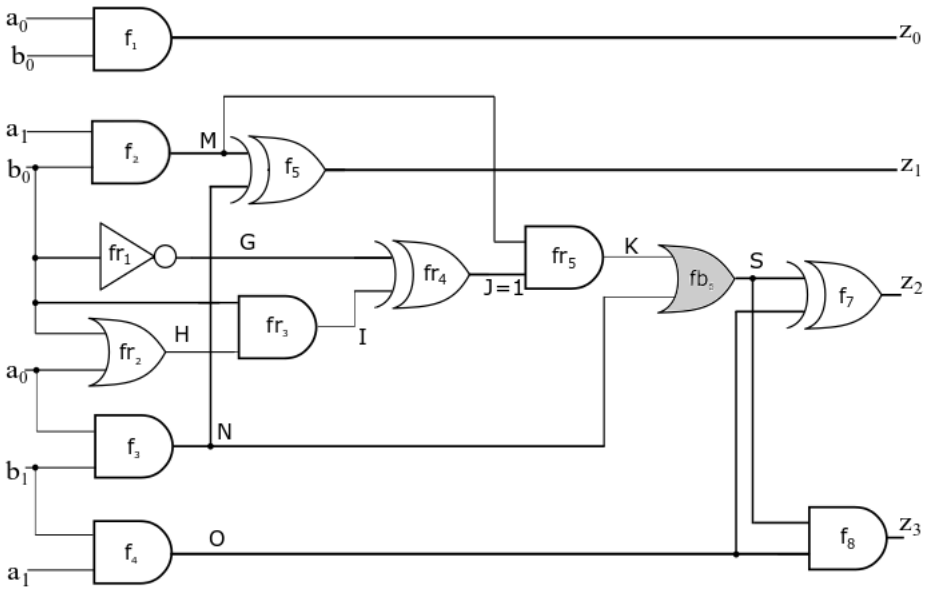

Figure 7.2: 2-bit integer multiplier with redundancy. Bug at s.

We apply our procedure to compute a rectification polynomial that can be implemented at net s to rectify the circuit. We compute the polynomial $s = a_0 \cdot a_1 \cdot b_0, \cdot b_1$. *We now check if the polynomial can be written in terms of the nets* $\{k, n\}$. *We construct an ideal* $J = \langle f_1, \ldots, f_{b6} : s - a_0 \cdot a_1 \cdot b_0, \cdot b_1, \ldots, f_8$. *We also consider the ideal* $J_0 = \langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. *Consider the following term order, with s, k and n last in the term order.*

$lex : \{z_3 > z_2 > z_1 > z_0\} > \{j\} > \{i\} > \{m > o > g > h\} > \{a_0 > a_1 > b_0 > b_1\} > \{s\} > \{k\} > \{n\}$

The above order is an elimination order, used to eliminate all the variables except s, k and n.

We find the Gröbner Basis $GB(J + J_0)$. *This Gröbner Basis contains 26 polynomials. We find four polynomials in variables s, k and n. They are:*

$$n^2 - n$$

$$k^2 - k$$

$$\boldsymbol{s - k \cdot n}$$

$$b_1 \cdot n - n$$

We compute the polynomial $s - k \cdot n$, which is the rectification polynomial in terms of $\{k, n\}$.

In the above example, we computed a polynomial that can be written in terms of the given set X_{int}. However, this is not always the case. If we compute a polynomial that cannot be written in terms of the given set of internal nets, X_{int}, it is desirable to find another one which can be. Given a polynomial U which cannot be written in terms of variables X_{int}, our procedure cannot compute a rectification polynomial U' which can be written in terms of X_{int}. We propose further study of computing a rectification polynomial with constraints on the input variables it is dependent on.

7.2.3 Impact of term orders on efficiency

In section 6.4.1, we discuss the limitation of Gröbner Basis reduction with Reverse Topological Term Order (RTTO) imposed on the variables. We witness the problem of variable explosion. Consider the example below, where Gröbner Basis reduction is performed with a forward topology based term order.

Example 7.3. *Consider again the 2-bit multiplier with redundancy shown in Fig. 7.2. A bug is introduced at net s as shown in Example 7.2*

Perform verification using the RTTO order:

$lex : \{z_3 > z_2 > z_1 > z_0\} > \{s\} > \{k\} > \{j\} > \{i\} > \{m > n > o > g > h\} > \{a_0 > a_1 > b_0 > b_1\}$

The remainder we compute is $f \xrightarrow{J+J_0} -4 \cdot a_0 \cdot a_1 \cdot b_0 \cdot b_1 + 4 \cdot a_0 \cdot b_1$

Now let us perform verification using a different order based on the topology of the circuit. Reduction is performed using the forward order:

$lex : \{a_0 > a_1 > b_0 > b_1\} > \{m > n > o > g > h\} > \{i\} > \{j\} > \{k\} > \{s\} >$

$\{z_3 > z_2 > z_1 > z_0\}$

The remainder computed is:

$f \xrightarrow{GB(J+J_0)} 4 \cdot s - 4 \cdot z_3 \cdot z_0$

We see that forward topology based term orders compute smaller remainders, which can overcome the limitation of Gröbner Basis reduction. There may exist other term orders, like degree lexicographic order, that also reduce the complexity of Gröbner Basis reduction. However, the use of these term orders requires the computation of Gröbner Basis. Therefore, further research is needed in formulating the problem of single-fix rectification using a forward-topological term order efficiently.

7.2.4 Multi-fix rectification

Not all the bugs that may be present in a circuit can be rectified at a single location. Some bugs can be rectified only by the implementation of multiple functions at multiple nets in the circuit.

Example 7.4. *Consider the 2-bit integer multiplier shown in Fig.7.3. Bugs are introduced at nets I and z_0. AND gates at nets I and z_0 are replaced by XOR gates.*

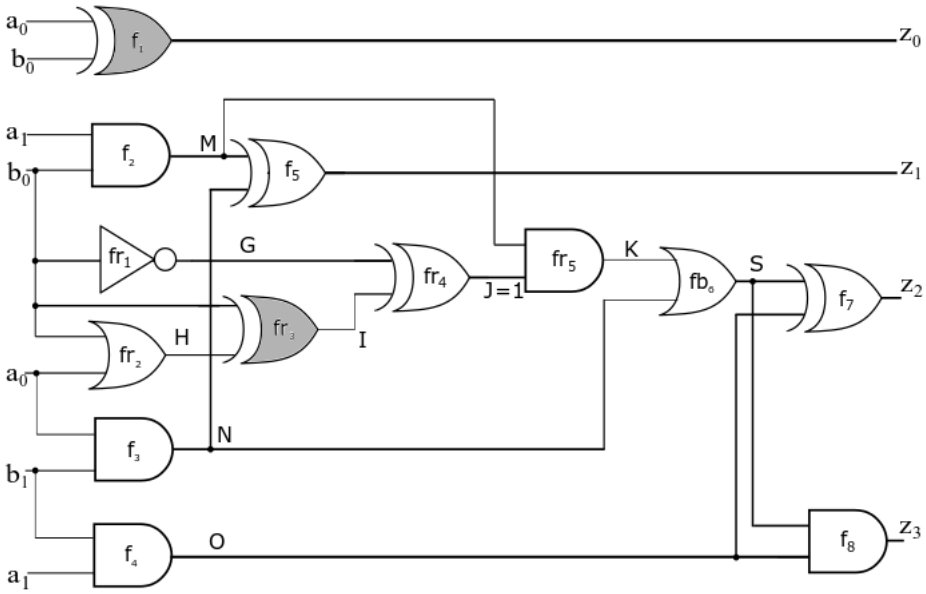

Figure 7.3: 2-bit integer multiplier with redundancy. Bug at I and z_0.

This multiplier cannot be rectified at a single location. Rectification of this circuit requires the implementation of at least two functions at different locations.

The problem of multi-fix rectification is a relevant problem that requires future study.

REFERENCES

[1] D. C. Black and J. Donovan, *SystemC: From the Ground Up*, Springer, second edition edition, October 2005.

[2] D. E. Thomas and P. R. Moorby, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, Boston, MA, second edition edition, 1994.

[3] D. L. Perry and H. F. Foster, *Applied Formal Verification*, McGraw-Hill Professional, first edition, April 2005.

[4] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Comp.*, vol. C-35, pp. 677–691, 1986.

[5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[6] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams", *in Proceedings of Design Automation Conference*, pp. 535–541, 1995.

[7] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, *Handbook of Model Checking*, Springer Publishing Company, Incorporated, 1st edition, 2018.

[8] T. Pruss, *Word-Level Abstraction from Combinational Circuits using Algebraic Geometry*, PhD thesis, University of Utah, May 2015.

[9] J. Lv, *Scalable Formal Verification of Finite Field Arithmetic Circuits using Computer Algebra Techniques*, PhD thesis, Univ. of Utah, Aug. 2012.

[10] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", *in Comp. Aid. Verif.*, vol. 6174, pp. 24–40, 2010.

[11] F. Lu, L. Wang, K. Cheng, and R. Huang, "A Circuit SAT Solver With Signal Correlation Guided Learning", *in IEEE Design, Automation and Test in Europe*, pp. 892–897, 2003.

[12] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to Combinational Equivalence Checking", *in Proc. Intl. Conf. on CAD (ICCAD)*, pp. 836–843, 2006.

[13] Daniela Kaufmann, Armin Biere, and Manuel Kauers, "Incremental column-wise verification of arithmetic circuits using computer algebra", *Formal Methods in System Design*, Feb 2019.

[14] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of Gate-level Arithmetic Circuits by Function Extraction", *in 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[15] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal Verification of Integer Multipliers by Combining Gröbner Basis with Logic Reduction", *in Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1048–1053, 2016.

[16] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths", *in Computer Aided Verification Conference*, pp. 473–486, 2008.

[17] N. Shekhar, P. Kalla, and F Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing", *IEEE Transactions on CAD*, vol. 26, pp. 1320–1330, July 2007.

[18] N. Shekhar, P. Kalla, M. B. Meredith, and F. Enescu, "Simulation Bounds for Equivalence Verification of Polynomial Datapaths using Finite Ring Algebra", *IEEE Transactions VLSI*, vol. 16, pp. 376–387, 2008.

[19] N. Tew, P. Kalla, N. Shekhar, and S. Gopalakrishnan, "Verification of Arithmetic Datapaths using Polynomial Function Models and Congruence Solving", *in Proc. Intl. Conf. on Computer-Aided Design (ICCAD)*, pp. 122–128, 2008.

[20] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits", *in IEEE Trans. on CAD*, vol. 32, pp. 1409–1420, 2013.

[21] T. Pruss, P. Kalla, and F. Enescu, "Efficient Symbolic Computation for Word-Level Abstraction from Combinational Circuits for Verification over Finite Fields", *IEEE Trans. on CAD*, vol. 35, pp. 1206–1218, July 2016.

[22] A. Lvov, L. Lastras-Montano, B. Trager, V. Paruthi, R. Shadowen, and A El-Zein, "Verification of Galois field based circuits by formal reasoning based on computational algebraic geometry", *Formal Methods in System Design*, vol. 45, pp. 189–212, Oct 2014.

[23] X. Sun, P. Kalla, T. Pruss, and F. Enescu, "Formal verification of sequential galois field arithmetic circuits using algebraic geometry", *in Proc. Design, Automation and Test in Europe*, 2015.

[24] X. Sun, P. Kalla, and F. Enescu, "Word-level Traversal of Finite State Machines using Algebraic Geometry", *in Proc. High-Level Design Validation and Test*, 2016.

[25] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, 2007.

[26] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*, American Mathematical Society, 1994.

[27] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design", *Proceedings of the IEEE*, 2015.

[28] M. Fujita and A. Mishchenko, "Logic Synthesis and Verification on Fixed Topology", *in 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, 2014.

[29] J. C. Madre, O. Coudert, and J. P. Billon, "Automating the Diagnosis and the Rectification of Design Errors with PRIAM", *in Proc. ICCAD*, pp. 30–33, 1989.

[30] H. T. Liaw, J. H. Tsaih, and C. S. Lin, "Efficient Automatic Diagnosis of Digital Circuits", *in Proc. ICCAD*, pp. 464–467, 1990.

[31] C. C. Lin, K. C. Chen, S. C. Chang, and M. Marek-Sadowska, "Logic Synthesis for Engineering Change", *in Proc. Design Automation Conf. (DAC)*, pp. 647–652, 1995.

[32] C. Scholl and B. Becker, "Checking Equivalence for Partial Implementations", *in Design Automation Conference (DAC)*, 2001.

[33] K. F. Tang, C. A. Wu, P. K. Huang, and C. Y. Huang, "Interpolation-Based Incremental ECO Synthesis for Multi-Error Logic Rectification", *in Proc. Design Automation Conf. (DAC)*, pp. 146–151, 2011.

[34] S. Jo, T. Matsumoto, and M. Fujita, "SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions", *in IEEE 21st Asian Test Symposium*, 2012.

[35] W. Craig, "Linear reasoning: A new form of the Herbrand-Gentzen theorem", *Journal of Symbolic Logic*, vol. 22, pp. 250–268, 1957.

[36] B. H. Wu, C. J. Yang, C. Y. Huang, and J. H. R. Jiang, "A Robust Functional ECO Engine by SAT Proof Minimization and Interpolation Techniques", *in Intl. Conf. on Comp. Aided Des.*, pp. 729–734, 2010.

[37] A. C. Ling, S. D. Brown, S. Safarpour, and J. Zhu, "Toward Automated ECOs in FPGAs", *IEEE Trans. CAD*, vol. 30, pp. 18–30, 2011.

[38] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence Checking of Partial Designs Using Dependency Quantified Boolean Formulae", *in IEEE International Conference on Computer Design (ICCD)*, 2013.

[39] C. Yu, X. Zhang, D. Liu, M. Ciesielski, and D. Holcomb, "Incremental SAT-Based Reverse Engineering of Camouflaged Logic Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[40] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging using Boolean Satisfiability", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[41] F. Farahmandi and P. Mishra, "Automated Debugging of Arithmetic Circuits Using Incremental Gröbner Basis Reduction", *in IEEE International Conference on Computer Design (ICCD)*, 2017.

[42] F. Farahmandi and P. Mishra, "Automated Test Generation for Debugging Arithmetic Circuits", *in Des. Auto. Test in Eur. Conf. (DATE)*, 2016.

[43] S. Ghandali, C. Yu, D. Liu, W. Brown, and M. Ciesielski, "Logic Debugging of Arithmetic Circuits", *in IEEE Computer Society Annual Symposium on VLSI*, 2015.

[44] U. Gupta, I. Ilioaea, V. Rao, A. Srinath, P. Kalla, and F. Enescu, "On the Rectifiability of Arithmetic Circuits using Craig Interpolants in Finite Fields", *in IFIP/IEEE International Conference on Very Large Scale Integration*, Oct 2018.

[45] U. Gupta, I. Ilioaea, P. Kalla, F. Enescu, V. Rao, and A. Srinath, "Craig Interpolants in Finite Fields using Algebraic Geometry: Theory and Applications", *in Intl. Workshop on Logic and Synthesis (IWLS)*, June 2018.

[46] V. Rao, U. Gupta, I. Ilioaea, A. Srinath, P. Kalla, and F. Enescu, "Post-verification debugging and rectification of finite field arithmetic circuits using computer algebra techniques", *in 2018 Formal Methods in Computer Aided Design (FMCAD)*, pp. 1–9, Oct 2018.

[47] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, PhD thesis, University of Innsbruck, 1965.

[48] Thomas W. Dubé, "The structure of polynomial ideals and grobner bases", *SIAM J. Comput.*, vol. 19, pp. 750–773, June 1990.

[49] Y. N. Lakshman, "On the complexity of computing a Gröbner basis for the radical of a zero dimensional ideal", *in Symp. on Theory of Computing (STOC)*, pp. 555–563, 1990.

[50] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra (Undergraduate Texts in Mathematics)*, Springer, July 2005.

[51] J. Lv, P. Kalla, and F. Enescu, "Efficient Groebner Basis Reductions for Formal Verification of Galois Field Multipliers", *in IEEE Design, Automation and Test in Europe*, 2012.

[52] N. Shekhar, *Equivalence Verification of Arithmetic Datapaths using Finite Ring Algebra*, PhD thesis, University of Utah, 2007.

[53] S. Gao, "Counting Zeros over Finite Fields with Gröbner Bases", Master's thesis, Carnegie Mellon University, 2009.

[54] B. Buchberger, "A criterion for detecting unnecessary reductions in the construction of a groebner bases", *in EUROSAM*, 1979.

[55] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Multipliers", *in Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 899–904, 2012.

[56] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann, "SINGULAR 4-1-0 — A computer algebra system for polynomial computations", `http://www.singular.uni-kl.de`, 2016.