12-16-2008

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

Eric D. Simonaire

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

THESIS

Eric D. Simonaire

AFIT/GCO/ENG/09-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCO/ENG/09-02

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Cyber Operations

Eric D. Simonaire, BS

16 December 2008

AFIT/GCO/ENG/09-02

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

Eric D. Simonaire, BS

Approved:

/signed/	5 Dec 2008
Lt Col J. Todd McDonald, Ph.D. (Chairman)	Date
/signed/	5 Dec 2008
Dr. Yong C. Kim (Member)	Date
/signed/	5 Dec 2008
Dr. Daniel W. Repperger (Member)	Date

AFIT/GCO/ENG/09-02

Abstract

Intent protection is a model of software obfuscation which, among other criteria, prevents an adversary from understanding the program's function for use with contextual information. Relating this framework for obfuscation to malware detection, if a malware detector can perfectly normalize a program P and any obfuscation (variant) of the program $O(P)$, the program is not intent protected. The problem of intent protection on programs can also be modeled as intent protection on combinational logic circuits. If a malware detector can perfectly normalize a circuit C and any obfuscation (variant) $O(C)$ of the circuit, the circuit is not intent protected.

In this effort, the research group set the primary goal as determining if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm, even when the transformation algorithm is known. The research group set the secondary goal as relating this result on circuit transformations to the realm of software obfuscation. The transformation rules of the sub-circuit selection and replacement algorithm are identified and modeled as rewrite rules in a term rewriting system. These rewrite rules are examined for critical overlaps which cannot be resolved by a widely used completion algorithm known as Knuth-Bendix. The research group performs an analysis of the critical overlaps found within the rewrite rules and successfully relates these results to the instruction-substitution obfuscations of a software obfuscator.

Acknowledgements

I would first like to thank the Lord for giving me the opportunity to attend the Air Force Institute of Technology and complete this Masters program. I would like to thank my parents for all their support and for encouraging me to finish my education as soon as possible. I would also like to thank my fiance for putting up with my I've-been-up-for-seventy-two-hours-straight-coding-o'clock shadow, as well as for supporting me from a few hundred miles away. I would like to thank my advisor and committee members for their guidance throughout the entire research effort as well as for introducing me to Lebanese food. I would also like to thank Arun Lakhotia for his insight and collaboration in this research area. Finally, I would like to thank Nate for letting me muse on and on about term rewriting systems each afternoon and Matt for pretending that all my scribblings on the whiteboard made any sense at all.

Eric D. Simonaire

Table of Contents

List of Figures

List of Tables

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

I. Introduction

1.1 Background

Metamorphic malware are programs which contain two components: a metamorphic engine and a malicious payload. Metamorphic engines using instruction-substitution obfuscations modify the instructions of a malware during replication causing new generations of the same malware to contain different segments of code. Walenstein *et al.* [14] provide several approaches malware detectors use to detect these metamorphic malware. One such approach is to map the instructions or syntax of a program to certain signatures which can be used to detect a malware. However, metamorphic malware modifies its own code during replication creating different signatures. In order for a malware detector to detect all possible variants of a malware, it must contain all possible signatures matching what a malware can become, a number which may quickly become unusable. A second approach given to detect metamorphic malware is pattern matching or more general signatures. Patterns may abstract out specific syntactical differences between signatures to match larger classes of metamorphic variants. However, the problem of creating patterns to match many variants is difficult and the number of patterns needed to match all possible variants may also become unusable.

Walenstein *et al.* [14] offer a third approach to malware detection: program normalization. They claim that program normalization removes unimportant variations between generations of metamorphic malware, and combined with pattern matching, may become an effective means of malware detection. Lakhotia *et al.* [6] developed a "generic" normalizer for C programs which, though it could not perfectly normalize malware, significantly reduced the number of variants of generic C programs. Walenstein *et al.* [14] then address the following research question: "When are perfect normalizers possible?" They define a restricted normalization problem and claim that perfect normalization is possible for some malware when the metamorphic engine is known.

Intent protection as defined by McDonald and Yasinsac [7] is a form of software obfuscation which, among other criteria, prevents an adversary from understanding the program's function for use with contextual information. If an adversary can perfectly normalize both a program P and an obfuscated version of the program $O(P)$ to one normal form, the adversary has identified $O(P)$ as P and assuming the adversary can understand the function of P, the adversary also understands the function of $O(P)$. Therefore, if a malware detector can perfectly normalize a program P and any obfuscation (variant) $O(P)$ of the program, the program is not intent protected.

McDonald and Yasinsac [7] then narrow the problem of intent protection to the obfuscation of combinational logic circuits and provide positive results in the realm of software obfuscation by modeling an instruction-substitution obfuscation algorithm as a sub-circuit selection and replacement algorithm. Based on the requirements of intent protection and modeling software obfuscation as the problem of circuit obfuscation, if a malware detector can perfectly normalize a circuit C and any obfuscation (variant) $O(C)$ of the circuit, the circuit is not intent protected.

1.2 Research Goals and Hypothesis

The primary goal of this research effort is to determine if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm if the transformation algorithm is previously known. This goal is met when the transforming rules of a sub-circuit selection and replacement algorithm are modeled as rewrite rules in term rewriting theory and it is determined if there exist critical overlaps within these rewrite rules that cannot be resolved thereby preventing this rule set from converging.

The secondary goal of this research effort is to determine the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and to determine their effectiveness in the realm of software obfuscation. This goal is met when the cause of critical overlaps within the rule set is identified and related to the obfuscating transformations of instruction-substitution algorithms. We hypothesis and test whether a malware detector, based upon the mechanisms of term rewriting theory, can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm, even when the transformation algorithm is known.

1.3 Document Overview

Chapter II provides an overview of relevant positive and negative results in the realm of software obfuscation as well as malware detection. Chapter III defines the methodology used in this research effort. Chapter IV presents the data collected as a result of exercising the experimental framework defined in Chapter III and gives us foundation to answer the questions posed by this research effort. Chapter V presents the conclusions of this research result and their significance as well as future areas of research.

II. Software Obfuscation and Metamorphic Malware

This chapter examines several positive and negative results related to the field of software obfuscation. Section 2.1 reviews well-known impossibility results of general obfuscation based upon the virutal black-box and best-possible obfuscation models. Section 2.2 summarizes an alternative model for obfuscation known as program encryption based upon the Random Program Model. Section 2.3 discusses past work on metrics and measures related to entropy and randomness. Section 2.4 relates the fields of software obfuscation with malware detection highlighting positive results in malware detection relevant to deobfuscation.

2.1 General Obfuscation

Barak *et al.* [2] provide a negative result proving that no 2-Turing Machine (2-TM) or 2-Circuit obfuscator exists. Informally, they define a obfuscator O as an efficient, probabilistic "compiler" which takes a program P as input and produces an obfuscated version of the program $O(P)$ as the output. They claim that an obfuscator must meet the following criteria:

1. **functionality**, which requires that $O(P)$ compute the same function as P,
2. **polynomial slowdown**, which requires that $O(P)$ is at most polynomially slower than P,
3. **virtual black-box (VBB) property**, which requires that any information which can be efficiently computed from $O(P)$ can also be computed given oracle access to P.

Barak *et al.* reach their impossibility result by constructing a family F of functions with the property $\pi : F \rightarrow 0, 1$ under the following conditions:

1. $\pi(f)$ can be efficiently computed given any program with a function $f \in F$,
2. Given oracle access to a randomly selected function $f \in F$, no efficient algorithm can compute $\pi(f)$ much better than by random guessing.

These conditions show that no general obfuscator (under the VBB security condition) exists for programs which compute these functions, as the obfuscator cannot hide $\pi(f)$. Therefore Barak *et al.* conclude that a different security condition, apart from the VBB property, must be presented in order to construct a general obfuscator.

In another study, Goldwasser and Rothblum [4] present a different notion of software obfuscation known as *best-possible obfuscation*. Best-possible obfuscation guarantees that whatever information is leaked by an obfuscated program $O(P)$, the same information is also leaked by any other program P which computes the same functionality. While this model of obfuscation provides no guarantee to hide any specific information in program P, it does guarantee that $O(P)$ is the best possible obfuscation of P.

1. **Indistinguishability Obfuscation.** An algorithm O which takes a circuit C as an input and outputs a new circuit is said to be a best-indistinguisability obfuscator for the family C, if it both preserves functionality and exhibits a polynomial slowdown along with the following property:

    - **Computationally/Statistically/Perfectly Indistinguishable Obfuscation**. For large input lengths, for any circuit $C_1 \in C_n$ and for any circuit $C_2 \in C_n$ that compute the same function as C_1 and $|C_1| = |C_2|$, $O(C_1)$ and $O(C_2)$ are computationally/statistically/perfectly indistinguishable.

2. **Best-Possible Obfuscation.** An algorithm O which takes a circuit C as an input and outputs a new circuit is said to be a best-possible obfuscator for the family C, if it both preserves functionality and exhibits a polynomial slowdown along with the following property:

    - **Computationally/Statistically/Perfectly Best-Possible Obfuscation**. For large input lengths, for any polynomial size circuit adversary A, there exists a polynomial size simulator circuit S such that for any circuit $C_1 \in C_n$ and for any circuit $C_2 \in C_n$ that compute the same function as C_1 and $|C_1| = |C_2|$, $A(O(C_1))$ and $S(C_2)$ are computationally/statistically/perfectly indistinguishable.

This definition guarantees that any information an adversary A can compute from $O(C_1)$ can also be computed from a simulator S on any program C_2 of the same size and function.

Goldwasser and Rothblum further prove that if O is an efficient indistinguishability obfuscator for a program P, then it is also an efficient best-possible obfuscator for C. If Δ is the distance measure in the guarantee of the obfuscator, then for any two circuits C_1 and C_2 of the same size and functionality, $\Delta(O(C_1), S(C_2)) \leq \epsilon$, and $\Delta(O(C_2), S(C_2)) \leq \epsilon$ therefore:

$$\Delta(O(C_1), O(C_2)) \leq 2\epsilon \tag{2.1}$$

Finally, Goldwasser and Rothblum prove that if the family of $3-CNF$ formulas can be statistically best-possible obfuscated, even in non-polynomial time, then there is a collapse in the polynomial hierarchy.

2.2 Program Encryption

After proving that general obfuscators satisfying the functionality, polynomial slowdown, and VBB property do not exist, Barak *et al.* refer to the VBB property as "inherently flawed". McDonald [9] considers the two questions posed by Barak *et al.* in determining whether an alternative security property of obfuscation exists:

1. Are there weaker or alternative methods for obfuscation that provide meaningful results?
2. Can we construct obfuscators for restricted but non-trivial/interesting classes of programs?

Based upon these questions, [7–9] provide an alternative model of obfuscation and show that general obfuscators do exist in a random program model which are not subject to Barak's impossibility proof. The following definitions formalize the ideas of understandability, obfuscation, and intent protection in this model.

Definition 1. Black-Box Understandable/Obfuscated Program $P \rightarrow X, Y$ is black-box understandable if and only if given an arbitrarily large set of pairs $IO = x_i, y_i$ such that $y_i = P(x_i)$ and y_j an arbitrary element of Y (not an element of IO), an adversary can guess x_j such that $y_j = P(x_j)$ in polynomial time on the length of P with probability $> \epsilon$.

Definition 2. White-Box Understandable/Obfuscated, Informal Program P is white-box understandable if it is understandable through static or dynamic analysis of P or a collaboration of the two. Otherwise, we say P is white-box obfuscated.

Definition 3. Intent Protected Program P is intent protected if and only if it is black-box protected, white-box protected, and protected from any composition of the two.

McDonald and James [7] summarize three properties which form the basis of the majority of theoretical and practical models of obfuscation:

1. **Semantic Equivalence**. $\forall x \in 0, 1^n : P(x) = P'(x)$, where n is the input size of P and $P' = O(P)$.
2. **Efficiency**. There is a polynomial l such that for every circuit P, $|O(P)| \leq l(|P|)$.
3. **Security**. A property that expresses some notion of information "hiding" or security guaranteed by $O(\cdot)$ for every possible circuit under consideration. The expression and measurement of the property varies from model to model.

Considering these definitions and properties, [8] define a model of obfuscation. In order to make concrete statements applicable to software obfuscation, they claim that researchers have based general representations of programs as either Turing machines or circuits. McDonald and Yasinsac chose to define obfuscation transformations on circuits. [7] shows that they can simulate a Turing machine TM on inputs having length n with a single n-input circuit with size $O((|TM| + n + t(n))^2)$ where $t(n)$ bounds the running time of TM for inputs of length n. More precisely, they base their results on *combinational* logic circuits and subsequent references in this section to *circuits* refer to *combinational logic circuits*.

In order for a program P to be intent protected, P must be black-box protected, white-box protected, and protected from any composition of the two. In order to achieve a useful black-box transformation, McDonald and Yasinsac provide the following two requirements:

1. *Change in Black-Box Behavior.* The functional behavior changes for some majority of the values in the domain x, $P(x) \neq P'(x)$.
2. *Recovery of Black-Box Behavior.* In order to recover the original functional output of P, some function $S(\cdot)$ must allow inversion: $\forall(x) : P(x) = S(P'(x))$.

Following these two requirments, McDonald and Yasinsac provide two black-box transformations which achieve stronger guarantees of security, *black-box refinement* and *semantic transformation*. They refer to black-box refinement as any of the following modifications to a circuit:

1. Adding a random number of input bits
2. Randomly permuting the input bits
3. Introducing intermediate gates which take inputs from each of the new gates and some random number of the original input signals of P
4. Adding a random number of output bits
5. Randomly permuting the output bits

McDonald and Yasinsac refer to semantic transformations as transformations which compose a circuit with a semantically strong encryption algorithm. The algorithm $t(p, k) = (p', r)$ is a process that creates a circuit p' so that it has a strongly one-way input/output relationship with an original circuit p. While there may be many possible semantic transformations possible, they explore transformations which compose the output of the original circuit p with the input of a strong data encryption circuit. This procedure is illustrated in figure 1.

Finally, intent protection also requires the most traditional form of obfuscation, white-box obfuscation. A white-box transformation $w(p, k) = p'$ takes as input a circuit p and some information embodied in a key k producing a circuit p' which is a functionally equivalent yet more confused variant of p. McDonald and Yasinsac state that while there are possibly an unlimited number of white-box obfuscation algorithms, they have implemented an algorithm based on sub-circuit selection and replacement. The algorithm selects a small $(1 - 5$ gates) candidate subcircuit with i inputs, o outputs, and computes its truth table TT. The algorithm then uniformly and randomly selects a replacement circuit from the set of circuits with i inputs, o outputs, and TT truth table. The algorithm is run iteratively until the security property of intent protection is satisfied.

As Barak *et al.* have shown that no general obfuscator exists under the VBB model, Yasinsac and McDonald [15] provide an alternative model known as the Random Program Model. Under this model, a random program oracle transforms any program P into an alternate version P'. After an adversary knows any n pairs of original and encrypted programs $\{(P_1, P_1'), (P_2, P_2'), (P_{n-1}, P_{n-1}'), (P_n, P_n')\}$ and supplies a program P_{n+1}, the adversary will receive P_{n+1}' which is either: a random program (P_R) or the obfuscated version of the program $O(P_{n+1})$. The program $O(P)$ provides intent protection if and only if the probability that an adversary is able to distinguish the obfuscated version (P_{n+1}') from a random program (P_R) is $\frac{1}{2} + \epsilon$ where ϵ is negligible.

2.3 Metrics Relating to Random Programs

This section will now consider additional related works on metrics of entropy and randomness related to circuits. Rajgopal [12] presents spatial entropy as an information theoretic basis metric. According to [12], the information theoretic definition of entropy is a measure of information content in a system, which he says can be viewed as the measure of disorder in a system. He then defines spatial entropy as the measure of spatial disorder in a system which captures the spatial distance between inputs and outputs in a system. As a system computes data, data is propogating from the inputs to the outputs thus reducing the spatial disorder (entropy) in the system.

Rajgopal then defines spatial entropy relating to circuits. As spatial entropy is the communication effort required to compute the circuit, both gates and wires contribute to this effort. Gates compute Boolean values and wires propogate these values. He notes that while it is the wires that determine how the bits travel across the circuit, the gates determine the distribution of Boolean values and together one can measure the dynamic communication effort required in the circuit.

Rajgopal defines a circuit as a directed weighted graph $G = \langle V, E, L \rangle$ where each primary input, primary output, and logic gate are represented by a node $v \in V$ and each wire is represented as an edge $(v, w) \in E$ with a length attribute $l_{(v,w)} \in L$ which is the wire length. $L : E \rightarrow \Re$ where $\Re$ is the set of real numbers. v is the source node and w is the destination node for each edge.

Rajgopal provides the classical entropy function defined in information theory:

$$H(p_i) = \sum_{i=1}^{N} p_i \log(\frac{1}{p_i}) \tag{2.2}$$

N is the total number of possible values in a given system.

The following is the equation to compute the distribution of the Boolean values computed at node w by the binary entropy function $H(p_w^1, p_w^0)$:

$$H(p_w^1, p_w^0) = p_w^0 \log(\frac{1}{p_w^0}) + p_w^1 \log(\frac{1}{p_w^1}) \tag{2.3}$$

Rajgopal now defines the spatial entropy S for a circuit:

- The spatial entropy S at the output node of a single output circuit is the information-distance product over all the nodes in the circuit.

$$S = \sum_{v \in V} \sum_{w \in V} H_v * l_{(v,w)} \tag{2.4}$$

- H_v is the information computed at the node v over its input probability distribution and $l_{(v,w)}$ is the length of the fanout edge $(v, w) \in E$ from node v to node w.

  The spatial entropy of a multi-output circuit is defined as follows:

$$S = \sum_{i=1}^{m} S_{o_i} \tag{2.5}$$

m is the number of outputs and S_{o_i} is the spatial entropy at output o_i.

In order to compute spatial entropy for individual gate types, (assuming *unit edge length*) Rajgopal next defines local spatial entropy at a gate node $g \in V$ as:

$$\delta S_g = \sum_{g' \in V} H_g * l_{g,g'} \tag{2.6}$$

H_g is the information computed at the gate node g and $l_{(g,g')}$ is the length of the fanout edge from node g to node g'. As Rajgopal assumed $l_{(g,g')}$, this equation is now an approximation $\delta S_g = H_g$.

For example, a 2-input AND gate with the 1-probabilities p_x^1, p_y^1 at its inputs x, y has a 1-probability of $p_a^1 nd = p_x^1 * p_y^1$ as the only event yielding an output of 1 is $p_x^1 * p_y^1$. The local spatial entropy of a 2-input AND gate, δS_{and}, is the following:

$$H_{and} = p_{and}^0 \log \frac{1}{p_{and}^0} + p_{and}^1 \log \frac{1}{p_{and}^1} \tag{2.7}$$

Finally, as a second example a 2-input XOR gate with the 1-probability $p^1_{xor} = p^1_y(1 - p^1_x) + p^1_x(1 - p^1_y)$ as there is a one at the outputs with inputs $p^1_x * p^0_y$ or $p^0_x * p^1_y$. Therefore, the local spatial entropy of a 2-input XOR gate, δS_{xor}, is the following:

$$H_{xor} = p^0_{xor} \log \frac{1}{p^0_{xor}} + p^1_{xor} \log \frac{1}{p^1_{and}} \tag{2.8}$$

McDonald [15] refers to the properties of confusion and diffusion as being useful measures of intent protection under the Random Program Model. As with data encryption, the program encryption techniques must confuse or scramble the original program statements. Common implementations of confusion include selection and replacement algorithms. Not only must the program statements be confused, but they must also be distributed across the original program with operations that move confused code unpredictably, known as diffision.

2.4 Metamorphic Malware and Software Obfuscation

As shown by Dalla Preda *et al.* [10, 11], the field of malware detection is closely related to the field of software obfuscation. Dalla Preda discusses software piracy, malicious reverse engineering, and software tampering as known attacks that one attacker can use to gain an advantage over another. While software developers may rely on legal measures (copyrights, patents, and licenses) to protect their software, software obfuscation is an attractive technical solution.

Dalla Preda defines an obfuscator as a program which transforms programs in a way that the obfuscated code is functionally equivalent to the original code yet more difficult to understand. She also states that any attacker who has enough time, effort, and determination can reverse engineer any application and that the goal of software obfuscation is to delay the release of "confidential information" for a sufficient time.

Dalla Preda then shows that advances made in the field of software obfuscation closely relate to the field of malware detection. While there are many different forms of malware (viruses, worms, trojan horses, back-doors, and spyware) there are two major approaches to malware detection, *anomaly detection* and *misuse detection*. Anomaly detection assumes that behaviours of malicious code will differ from those normally observed on a system. While this approach has the advantage that no specific knowledge of a malware is required to detect an attack, it has the disadvantage that not all abnormal behaviours are malicious. Conversely, misuse detection, also known as signature or pattern-based detection, detects attacks by searching for patterns of known malware. While the disadvantages of misuse detection include the fact this system is not able to detect new attacks; the advantages are a low false positive rate and ease of use.

As malware writers attempt to avoid detection by these systems, obfuscated malware are becoming more prevalent. Dalla Preda defines two forms of obfuscated malware:

1. **Polymorphic malware.** Malware which changes its syntatic representation by encrypting its payload and decrypting during execution. This form of malware can be detected by techniques such as running it on a virtual system and observing its runtime behavior. As all forms of polymorphic malware look alike after decryption, misuse detection systems can be used.
2. **Metamorphic malware.** Malware which changes the syntax of each successive generation while leaving the semantics unchanged. The important point is that obfuscating transformations can easily defeat misuse detection systems. In order to detect metamorphic malware, standard misuse detection systems would have to keep a signature for all possible (which could be an unlimited number) mutations of the malware.

This background provides us a relationship between software obfuscation and metamorphic malware. Let us define the goal of software obfuscation as intent protection (described in section 2.2). Let us also define the goal of metamorphic malware as detection avoidance. If a program is intent protected, that is to say that it is black-box obfuscated, white-box obfuscated, and a protected against any composition of the two, then it also satisfies the property of detection avoidance. Conversely, if an intrusion detection system is able to detect a obfuscated metamorphic malware, then this malware is not intent protected as either black-box information, white-box information, or information from a combination of the two was leaked. This observation makes the study of advanced metamorphic malware detectors interesting as it may aid in defining useful and secure metrics for software obfuscation.

Walenstein *et al.* [14] claim that using term rewriting theory, they are able to provide approximate solutions to metamorphic malware detection. The approach the authors take to detecting metamorphic generations of malware is to normalize the malware in order to remove the changes that defeat misuse detection systems. They argue that the “perfect” normalizer would transform all variants of a specific malware to one normal form and call the problem of creating a normalizer for a specific metamorphic malware the “normalizer construction problem” (NCP).

Walenstein *et al.* form a version of the NCP, which they term “NCP=”, using term rewriting theory which is restricted by the following conditions:

1. An accurate model of the metamorphic engine is represented as a term rewriting system TRS
2. The metamorphic engine makes only semantic-preserving transformations

They show that while NCP= is undecidable (no procedure can exist which is guaranteed to halt and produce a correct normalizing transformation), approximations exist which are successful on certain interesting classes of programs. The approximations which they suggest are: (1) using “incomplete” rule sets, (2) using a priority scheme, and (3) ignoring conditions in the rule set.

Previous to this work, Lakhotia *et al.* [6] developed a *C* program normalizer which did not require a model of the metamorphic engine. This normalizer was able to remove transformations such as expression reshaping and constant propogation, as well as impose variable renaming, variable reordering, and instruction reordering. While their approach was not able to reduce general *C* programs to a normal form (due to transformations such as equivalent instruction substitutions), they did report a large reduction in the total number of possible normalized forms.

Walenstein *et al.* then provide different reduction stratagies in order to obtain a reduced form of a metamorphic program. If P is the metamorphic program, M is the metamorphic engine of P, $T = m_1, m_2, ..., m_3$ is the set of transfomations performed upon P, then $S(P)$ is the set of all possible variants of P that can be produced through the transformations of M. It follows then that if one knows M, then one *naive* approach would be to "reverse" the rules to produce a normalization. For example, if one transformation is $A \rightarrow B$ (statements A are transformed into statements B), then perhaps reversing the rule and applying $B \rightarrow A$ would correctly normalize the program. However, this strategy is not sufficient as the system is not guaranteed to follow the correct reversal of T and a different strategy, i.e. a TRS, is needed.

The following is a brief summation of the definitions of term rewriting theory which Walenstein *et al.* provide, though more detail can be found in [1, 14].

- **Terms, subterms, atomic, and ground.** *Terms* are constants, variables, functions, or functions on terms. A term t may contain other terms known as *subterms* of t. An *atomic* term does not contain any subterms. A *ground* term does not contain variables.
- **Term rewriting system (TRS).** A *TRS* is a set of *rewrite rules*, $s \rightarrow t$. Rewrite rules may be conditional, denoted by $p|R$ where rule R is to be applied only when condition p is true.
- **Reduction relation** $(\rightarrow_T)$**.** Given terms s and t, $(\rightarrow_T)$ is defined as follows: $s \rightarrow_T t$ holds iff for some rewrite rule $s' \rightarrow t'$, s has, as a subterm, an instance of s' which if replaced with it's corresponding instance of t', turns s into t.
- **Equivalence relation** $(\overset{*}{\leftrightarrow})$**.** The $\rightarrow$ relation on terms induces an equivalence relation $(\overset{*}{\leftrightarrow})$ defined by the reflexive symmetric transitive closure of $\rightarrow$. $(\overset{*}{\leftrightarrow})$

partitions the set of terms into equivalence classes. $[t]_T$ denotes the equivalence class of term t under $(\overset{*}{\leftrightarrow})$.

- **Normal form.** A term t is in *normal form* if it is not related to any other term under $\rightarrow_T$. $Norm_T(x)$ is the set of terms $[x]_T$ which are in normal form.

- **Termination.** A TRS T terminates if there exists no infinite chains of reductions $(t_1 \rightarrow t_2 \rightarrow t_3 ...)$.

- **Confluence.** If x, y, and z are arbitrary terms and there is a sequence of rules such that $x \rightarrow y$ and $x \rightarrow z$, then the system is confluent if every such y and z are *joinable.* Two terms y and z are joinable if there exists a set of rewrite rules such that y and z reduce to some arbitrary term w. The problem of converting an arbitrary TRS into an equivalent one that is confluent is undecidable [14].

- **Convergence.** A TRS is *convergent* if it is confluent and terminating. A convergent TRS T can be used to determine membership in any of the equivalence classes defined by $\overset{*}{\leftrightarrow}$ by applying the rules of T in any arbitrary order to any given input x. This process guarantees a unique normal form unique to x's equivalence class.

Therefore, if a TRS T is convergent, then given any variant of a program P, T is guaranteed to extract a unique normal form which will match any other variant of P.

As previously mentioned, Walenstein *et al.* show NCP= to be undecidable, though they define procedures which attempt to solve the problem. One procedure involves two phases: reorientation and completion. The reorientation phase reverses a rule's application direction and assigns orientations of the rules such that the reduction procedure is guaranteed to terminate by imposing some reduction order on terms [1, 14]. One frequently used reduction order used in term rewriting systems is the *well-founded length-lexicographic* ordering. This reduction order reorients rule in M whose right hand sides are length-lexicographically greater than their left hand sides. Unless there are rules of the form $x \rightarrow x$ then the resulting system M^t is terminating because any rule application decreases the length-lexicographic size of the reduced term. The reader is referred to [14] for detailed examples of reorientation.

After the procedure solving NCP= reorients the TRS, it must then complete the rule set. Walenstein *et al.* state again that while completing a TRS is in general undecidable, algorithms exist which attempt the completion. They select the Knuth-Bendix (KB) completion procedure to use in their examples as it is the most prevalent method used in term rewriting theory. The KB completion algorithm essentially works to resolve *critical overlaps* by adding certain rules. A critical overlap occurs either when the suffix of one term x on the right hand side is identical to the prefix of another term y or when one term s on the right hand side is a prefix of another term t. In these cases, the KB completion algorithm attempts to resolve this conflict by adding rules which eventually drive the reduction to the same term, independant of which rule is selected. A detailed explanation of the algorithm can be found in [5].

An important observation that Walenstein *et al.* make is that while a metamorphic engine which contained (non-preserving) semantic transformations may still be modeled as a TRS, doing so may make it difficult to reason about the problem of malware detection. They provide as an example a metamorphic engine with a rule $P \to B$ where P is the entire program and B is a known benign program. According to this ruleset, with respect to $[t]_M+$, B is equivalent to the original malware P since there is a rule which makes them equivalent. In the practical world, this scenario would introduce false positives into the TRS. Conversely, if the rule set is semantic preserving, a perfect normalized form is both complete and sound.

Finally, as completing a TRS in general is undecidable, Walenstein *et al.* provide several approximations for malware detection using a TRS T. The first approximation considered is if the completion procedure on T does not complete or is too large for normalization purpose, a non-completed (and non-confluent) rule set may be used. With this approximation, normalizers without a complete rule set may not reduce all variants of a malware P to one unique normal form, yet they will reduce all variants of P to a set of normal forms. The size of this set depends on the specifics of the non-completed ruleset and P. It is important to note that while variants of P may reduce to different normal forms, once again, no version of another program Q will reduce to any of these forms.

2.5 Background Summary

Several authors have published negative proofs for general obfuscation. Barak *et al.* prove that no general obfuscator exists which satisfies the VBB property. Goldwasser and Rothblum also prove that no best possible general obfuscators exist. Conversely, McDonald and Yasinsac provide alternate definitions of obfuscation which are not subject to the VBB property. They provide a model for obfuscation known as the Random Program Model and provide a test for general obfuscation within that model. Several metrics, such as spatial entropy, confusion, and diffusion have been related to circuits and may be useful in measuring circuit obfuscation. Dalla Preda relates the fields of software obfuscation and malware detection and provides examples of different forms of obfuscated malware. Finally, Walenstein *et al.* define a malware detector based upon the theory of term rewriting which may be able to perfectly normalize some forms of metamorphic malware.

III. Methodology

3.1 Problem Definition

The primary goal of this research effort is to determine if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm if the transformation algorithm is previously known. The research group meets this goal when the transforming rules of a sub-circuit selection and replacement algorithm are modeled as rewrite rules in term rewriting theory and the research group determines if there exist critical overlaps within these rewrite rules that cannot be resolved. If this reseach effort shows there exist rewrite rules which cannot be resolved, this will prevent a program normalizer from converging this rule set.

The secondary goal of this research effort is to determine the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and to relate their effectiveness to the realm of software obfuscation. The research group meets this goal when the cause of critical overlaps within the rule set is identified and related to the obfuscating transformations of instruction-substitution algorithms.

3.2 Approach

The approach this research group used to accomplish the primary goal is to model the sub-circuit selection algorithm as a malware detector based on a term rewriting system TRS and determine if there exist any critical overlaps within the transforming rules modeled as a rule set. A critical overlap occurs when the prefix of one rewrite rule in the TRS matches the suffix of another rule, or when one term in a rewrite rule is a subterm of another rewrite rule. If critical overlaps do exist, then the next step is to determine if a completion procedure is able to resolve the critical overlaps, creating a convergent rule set. As the problem of completing a TRS is undecidable in the general case [14], attempts to complete rule sets are not guaranteed to terminate. If the completion procedure is shown to produce a cycle, preventing termination of the algorithm, then it is shown that the rule set is non-convergent and therefore the transforming rules of the sub-circuit selection and replacement algorithm cannot be normalized using that completion procedure.

The approach used to accomplish the secondary goal is to utilize the malware detector modeling the sub-circuit selection and replacement algorithm in term rewriting theory and examine the factors contributing to the number of critical overlaps within the rule set. The approach is to then determine the relationship between rewrite rules utilized by a TRS and equivalent command substitution utilized by a software obfuscator and to draw relevant conclusions in the field of software obfuscation.

3.3 System Boundaries

To meet the goals of this research effort, the research group built the Circuit Transformation Analysis System CTAS. As shown in Figure 3.1, this system takes the following as inputs:

- **Circuit Generation Engine CGE**. An engine capable of producing circuits with I inputs, O outputs, and G gates which can be used by a sub-circuit selection and replacement algorithm.
- **Number of selected gates**. The number of selected gates N_{SG} used by the sub-circuit selection and replacement algorithm to create transformation rules.
- **Number of returned gates**. The number of returned gates N_{RG} used by the sub-circuit selection and replacement algorithm to create transformation rules.

Also shown in Figure 3.1, the CTAS provides the following two outputs:

- **Number of Rewrite Rules**. The number N_{RR} of rewrite rules found within the rule sets of the CGE.
- **Number of Critical Overlaps**. The number N_{CO} of critical overlaps found within the rewrite rules.

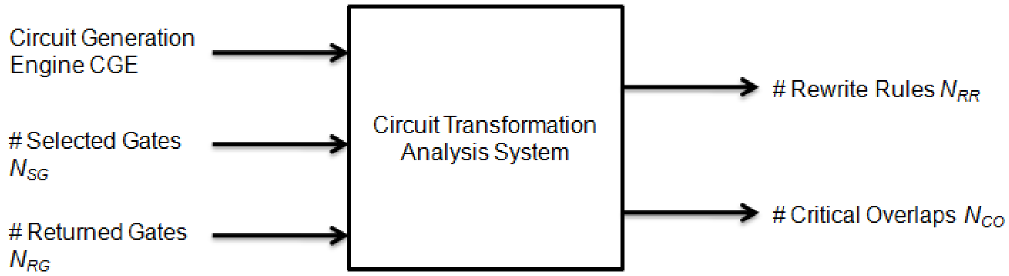

Figure 3.1: Circuit Transformation Analysis System

The CGE is a input of the sub-circuit selection and replacement algorithm. As shown in Figure 3.2, the CGE takes the following inputs:

- **Number of inputs**. The number of inputs I the generated circuits will contain. This input guarantees that when the CGE is utilized in a sub-circuit selection and replacement algorithm, the replacement circuits the CGE produces will contain the exact number of inputs necessary to properly replace the selected sub-circuit.
- **Number of outputs**. The number of outputs O the generated circuits will contain. This input guarantees that when the CGE is utilized in a sub-circuit selection and replacement algorithm, the replacement circuits the CGE produces will contain the exact number of outputs necessary to properly replace the selected sub-circuit.
- **Number of gates**. The number of gates G the generated circuits will contain. This input guarantees that the CGE will only produce sub-circuits which contain G gates.

As shown in Figure 3.2, the CGE provides the following as an output:

- **List of circuits**. This is a list of circuits L with I number of inputs, O number of outputs, and G number of gates.

It is important to note that while all sub-circuits generated by the CGE contain I inputs, not all sub-circuits utilize each of the I inputs. If these types of circuits are

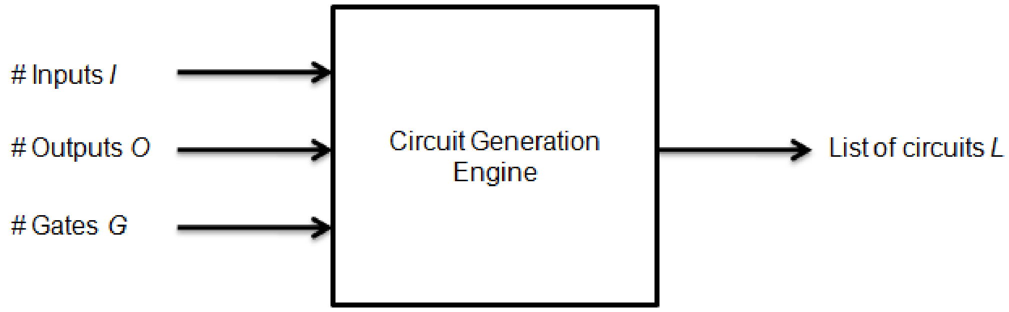

Figure 3.2: Circuit Generation Engine

Table 3.1: The signature of this truth table is the value of the output column O_1.

I_1	I_2	O_1
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

then returned as replacements in a sub-circuit selection and replacement algorithm, it may introduce intermediate gates whose output is neither an output of the circuit nor an input of any gate, otherwise known as a dangling gate.

A sub-circuit selection and replacement algorithm may use a CGE to generate replacement sub-circuits, which either preserve or transform semantics, for selected sub-circuits. If a sub-circuit selection and replacement algorithm selects sub-circuits containing G_1 gates and replaces them with sub-circuits containing G_2 gates, the sub-circuit selection and replacement algorithm will determine the input parameters I and O of the CGE based upon the number of inputs and outputs of the selected sub-circuit and it will initialize the input parameter G of the CGE to the value G_2. The CGE will then return the list of sub-circuits L which contain I inputs, O outputs, and G_2 gates. The sub-circuit selection and replacement algorithm can then choose a replacement sub-circuit from L. If the sub-circuit selection and replacement algorithm only considers semantic preserving transformations, then the algorithm must choose a replacement from L which contains the same signature, or output values, as the selected circuit.

3.4 Evaluation Technique

3.4.1 Enumerate Transformation Rules. The CTAS provides results to meet the goals of this research effort by performing measurements on a CGE used by a sub-circuit selection algorithm. In order to meet the first research goal, the CTAS first determines the total number of critical overlaps between rewrite rules in a malware detector using the transformation rules of a sub-circuit selection and replacement algorithm in each iteration.

In order to accomplish this first step, the CTAS enumerates all possible transforming rules a sub-circuit selection algorithm using a CGE may apply to a circuit. In this research effort, two sets of transformation rules are evaluated. During the first iteration, the CTAS examines all possible transformation rules that a sub-circuit selection and replacement algorithm may contain when selecting sub-circuits containing 1 gate and replacing them with sub-circuits containing 2 gates. During the second iteration, the CTAS examines all possible transformation rules when selecting sub-circuits containing 2 gates and replacing them with sub-circuits containing 3 gates.

In order to enumerate all possible transformation rules during each iteration, the CTAS determines the maximum number of possible unique inputs I and outputs O when selecting N_{SG} two-input gates. The maximum number of possible inputs of a selected sub-circuit with N_{SG} is the following:

$$N_I = 2 * N_{SG} \tag{3.1}$$

For example, the CTAS determines the maximum number of inputs a sub-circuit containing 3 two-input gates is 6. The maximum number of possible unique outputs of a selected sub-circuit with N_{SG} is the following:

$$N_O = N_{SG} \tag{3.2}$$

For example, the maximum number of outputs a sub-circuit containing 2 gates is 2.

As previously stated, the research group exerices the CTAS twice. During both executions, the CGE input remains the same and the I and the O inputs change. During a single execution, the CTAS iteratively executes the CGE with the input parameter I ranging from 1 to N_I, the output parameter O ranging from 1 to N_O, and the gates parameter G set to both N_{SG} and N_{RG}. During each iteration, the CTAS stores two lists of circuits. The first list, the list of selected gates L_S, is the list of gates generated by the CGE which contain N_{SG} gates. The second list, the list of replacement gates L_R, is the list of gates generated by the CGE which contain N_{RG} gates. Each circuit is stored with its signature S and the circuits stored in the list of selected gates L_S are also stored with a boolean value B, indicating the ability of a circuit to be selected within that circuit family.

The CTAS sets the value B to *true* when the circuit utilizes each of its inputs and the circuit is of size N_{SG}; the CTAS sets B to *false* otherwise. It is possible for the CGE to return circuits which do not use all of their possible inputs. These circuits are not valid sub-circuits which could be selected within the i-o-g family of circuits. It is necessary for the CTAS to determine this boolean value in order to determine the set of transformation rules.

After the CTAS creates the lists of selected and replacement circuits containing all possible circuits with up to N_I inputs and N_O outputs, and N_{SG} and N_{RG} gates, their signatures, and a boolean value indicating the ability to be selected, the CTAS is then able to create the list of transformations rules L_{TR}. A transformation rule R contains two variables: a circuit which can be selected C_S and a replacement circuit C_R. For each circuit C_i in the selected circuit list L_S which can be selected (B is set to *true*), the CTAS creates a rule for each circuit C_j in the replacement circuit list L_R where the signatures of C_i and C_j are equal, such that the selected circuit C_S is

Table 3.2: An example of a transformation rule in which the circuit C_S can be transformed to the circuit C_R.

Selected Circuit C_S	Replacement Circuit C_R
`0 = AND(-1, -2)`	`0 = AND(-1, -2)` `1 = AND(0, 0)`

Table 3.3: An example of a reduction relation in which the circuit l_i can be rewritten as the circuit r_i in a TRS.

		Rule			
Label	**Condition**	l_i	$\rightarrow$	r_i	**Reorient?**
M_i		`0 = AND(-1, -2)`	$\rightarrow$	`0 = AND(-1, -2)` `1 = AND(0, 0)`	y

C_i and the replacement circuit C_R is C_j. The CTAS then adds each of these rules to the list of transformation rules L_{TR}.

The list of transformation rules L_{TR} is now the complete list of semantic-preserving transformation rules possible which can be used in a sub-circuit selection and replacement algorithm using the CGE provided to the CTAS. Once this list is created, the first step of accomplishing the first goal of this research effort is accomplished.

3.4.2 Model Transformation Rules as a TRS. The second step of accomplishing the goals of this research is to model the transformation rules generated by a sub-circuit selection and replacement algorithm as a TRS. In order to accomplish this step, the transformation rules must be represented as rewrite rules within a TRS. Table 5 gives an example of a transformation rule with a 1 gate circuit being replaced by a functionally equivalent 2 gate circuit. The transformation rule from figure 5 can then be represented as a rewrite rules described in [14] as shown in figure 6.

As displayed in figure 6, the transformation rule of figure 5 maps into a rewrite rule of a term rewriting system. More specifically, I_1, I_2, O_1, and 1 are equivalent to variables, and both $=$ and *AND()* are equivalent to functions on terms. The *Label* in a rewrite rule is simply a unique identifier for each transformation rule. The *Condition* for each rewrite rule of a sub-circuit selection and replacement algorithm is always empty because there are no conditions checked (such as checking for live registers) when substituting equivalent sub-circuits. And finally, *Reorient* is set to *yes* if the number of gates on the left hand side (selectable gates) is less than the number of gates on the right hand side and it is set to *no* otherwise. As each transformation rule can be represented as a rewrite rule, once the CTAS has created the list of transfomation rules L_{TR}, it can then view these transformation rules as a set of rewrite rules and the second step is complete.

3.4.3 Reversing the Rule Set. The third step in determining if the rule set can be normalized is to create a malware detector M^t based upon the reoriented (reversed) rules of L_R. Rules can be reoriented by reversing the application direction. In the original rule set, terms on the left hand side (the sub-circuits which could be selected) could be rewritten as terms on the right hand side (replacement circuits). In the reoriented rule set, M^t, terms on the right hand side can be rewritten as terms on the left hand side, thereby reversing the sub-circuit selection and replacement transformations. This step is trivially accomplished by acknowledging that the replacement sub-circuits C_R now function as circuits which can be selected, and the selected sub-circuits now function as sub-circuits which can be replacements; thereby reversing the rule set.

3.4.4 Counting Critical Overlaps. For the fourth step to accomplish the goals of this research effort, the research group determines if there exist any critical overlaps as described in section 2.4. According to [14], a critical overlap occurs when the prefix of a rule x in the replacement sub-circuits matches the suffix of a rule y in the replacement circuits or when the suffix of a rule s in the replacement circuits matches the prefix of a rule t in the replacement circuits.

In order to determine the number of critical overlaps, the CTAS takes the prefix of each replacement circuit C_R within the list of transformation rules L_{TR} and records the replacement circuits, whose suffixes match the prefix, within a list of conflicting rules L_C. The prefix of a circuit represented as term in term rewriting theory is the first gate listed within that circuit. The suffix of a circuit represented as a term is the last gate of the circuit. The prefix and suffix match if the terms are equivalent. The fourth step is accomplished once the CTAS has created a list of conflicting rules L_C.

3.4.5 Completing the Rule Set. If the CTAS determines at this point that there are no critical overlaps, then the rule set is convergent and a perfect normalizer for the sub-circuit selection and replacement algorithm exists. If the CTAS does contain critical overlaps, then the rule set is not convergent and a perfect normalizer does not yet exist. However, completion procedures, such as the widely used Knuth Bendix completion procedure, are algorithms which attempt to resolve critical overlaps by adding additional rules which may cause a TRS to become convergent. If the rule set can be completed by a completion procedure, then the system is convergent and a perfect normalizer for the sub-circuit selection and replacement algorithm exists.

According to [14] the problem of completing a TRS is in general undecidable and is not guaranteed to terminate. By adding additional rules to L_{TR} to resolve critical overlaps, a procedure may be creating additional critical overlaps in C_R. In some cases, a completion procedure enters into a cycle, preventing the procedure from converging the rule set.

However, this research group notes that even if one completion procedure, such as the Knuth-Bendix completion procedure, is unable to complete the rule set, this does not imply that no completion procedure is able to complete the rule set. It is possible to complete rule sets through several different methods, even adding rules ad-hoc. It is not correct to assume that the failure of one completion procedure implies that a rule set cannot be completed by any procedure, though it may be reasonable to discuss the complexity of such other algorithms.

The CTAS accomplishes the first goal by counting the set of critical overlaps which exist after a completion procedure terminates or enters into a cycle. If the procedure terminates, then there are no critical overlaps remaining within the rule set M^t, therefore M^t is convergent and a malware detector using the set of rules found within M^t is able to perfectly normalize any circuit C and any obfuscated form of the circuit $O(C)$ to one normal form thereby discovering the identity of the circuit.

Conversely, if the completion procedure enters into a cycle and fails to terminate, then the rule set is non-convergent and a perfect normalizer for the sub-circuit selection and replacement algorithm does not yet exist. However, it is important to realize that another completion procedure may exist which can complete the rule set. In this case, it is necessary to determine the cost of other completion procedures which may terminate to determine their effectiveness.

3.5 Methodology Summary

In this effort, the research group set the primary goal as determining if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm if the transformation algorithm is previously known. The secondary goal is to determine the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and to determine their effectiveness in the realm of software obfuscation.

The approach is to model the sub-circuit selection and replacement algorithm in term rewriting theory and determine if there are irresolvable critical overlaps which prevent the transformation rule set from converging. If the transformation rule set is non-converging, then the causes should be identified and related to the realm of software obfuscation.

The Circuit Transformation Analysis System is the system built to accomplish the goals of this research interest. The CTAS takes the Circuit Generation Engine as an input and computes the transformation rule set. The CTAS then models this rule set as a set of rewrite rules in a term rewriting system and determines if there exist any critical overlaps. If there exist irresolvable critical overlaps which prevent the transformation set from converging, the CTAS provides the total number of critical overlaps. The causes of these critical overlaps should be identified and related to the realm of software obfuscation.

IV. Analysis and Results

4.1 Chapter Overview

In this chapter, we present and interpret the results of the research effort outlined in Chapter III. As a primary goal of this research effort, this research group determines if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm. To fulfill the secondary goal of this research effort, this reserach group determines the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and determine their effectiveness in software obfuscation. The research group represents the data of this experiment in tabular and graph form and interprets the data to accomplish the research goals.

4.2 Capabilities of the CGE

During the experiments, the CTAS executes the CGE with the number of inputs I, the number of outputs O, and the number of gates G. Unless otherwise stated, all circuits generated by the CGE are created from the six gate basis $\Omega = \{$AND, NAND, NOR, NXOR, OR, XOR$\}$. This chapter contains data on the circuit families being enumerated in this experiment. However, Appendix B provides tables which contain the cardinality of each circuit family δ_{I-O-G} with up to ten inputs, five outputs, and eight gates.

4.3 Results of Experiments and Literature Comparison

4.3.1 1 Gate Selection with 2 Gate Replacement. During the first execution of the CTAS, as outlined in Chapter III, the research group sets the number of selected gates N_{SG} to 1 and the number of returned gates N_{RG} is set to 2. This will allow the CTAS to enumerate all possible transformation rules utilized by a sub-circuit selection and replacement algorithm selecting sub-circuits containing one gate and replacing them with functionally equivalent sub-circuits containing two gates.

Table 4.1: The count of all sub-circuits which are able to be used by a sub-circuit selection and replacement algorithm selecting sub-circuits containing 1 gate and replacing them with sub-circuits containing 2 gates.

	1 Gate	**2 Gates**
1 Input - 1 Output	6	72
2 Inputs - 1 Output	6	324
Subtotals	12	396

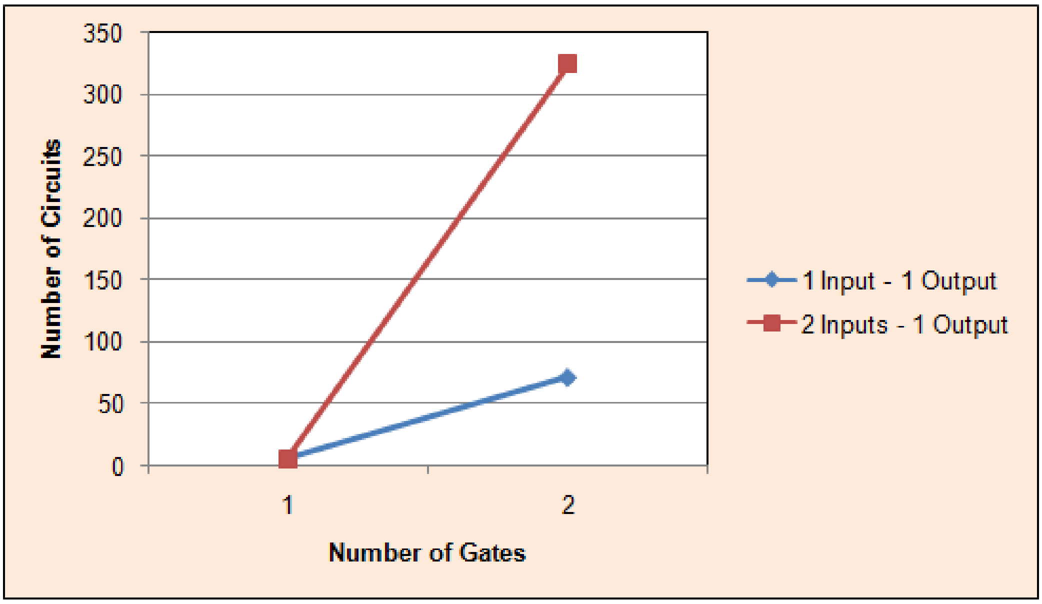

Figure 4.1: Number of Circuits (1 Gate Selection and 2 Gate Replacement)

In order to accomplish the first goal of the research effort, to determine if a malware detector can perfectly normalize the transformation rules of a sub-circuit selection and replacement algorithm which selects 1 gate and replaces it with 2 functionally equivalent gates, the CTAS first enumerates two lists. The first list L_S is the list of all possible 1 gate sub-circuits which can be selected within any circuit. The second list L_R is the list of all possible 2 gate sub-circuits which can be used for replacements. As shown in Table 4.1, the CTAS enumerates the list L_S of all possible sub-circuits, which can be selected within any circuit, containing only 1 gate and generates 12 unique sub-circuits. Also shown in Table 4.1, the CTAS enumerates the list L_R of all possible sub-circuits containing 2 gates and generates 396 possible replacements. Figure 4.1 provides a plot of these data.

Table 4.2: The count of all transformation rules possible in a sub-circuit selection and replacement algorithm selecting sub-circuits containing 1 gate and replacing them with sub-circuits containing 2 gates.

	1 Gate to 2 Gates
1 Input - 1 Output	104
2 Inputs - 1 Output	72
Total	176

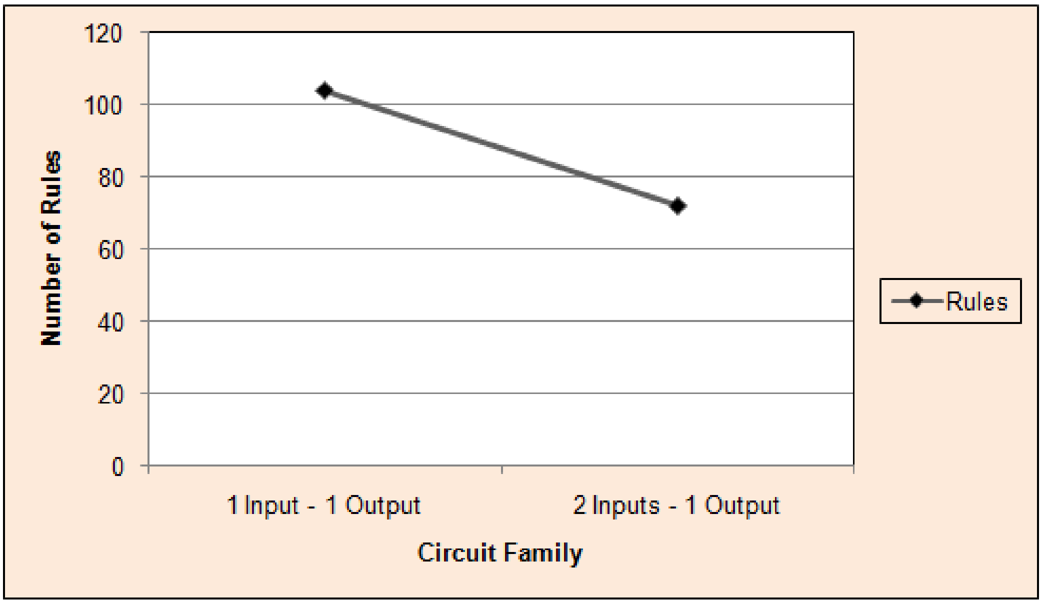

Figure 4.2: Number of Transformation Rules(1 Gate Selection and 2 Gate Replacement)

To accomplish the next step of the primary research goal, the research group determines all possible transformation rules for a sub-circuit selection and replacement algorithm selecting sub-circuits containing 1 gate and replacing them with sub-circuits containing 2 gates. As shown in Table 4.2, the CTAS enumerates all possible transformation rules from sub-circuits containing 1 gate to sub-circuits containing 2 gates and generates a total of 176 rules with a majority coming from the circuit family δ_{1-1}. Figure 4.2 provides a plot of this data.

The research group represents all 176 transformation rules generated by the CTAS as rewrite rules in term rewriting theory. Table 4.3 provides a subset of the

Table 4.3: A subset of the reduction relations used by a sub-circuit selection and replacement algorithm selecting sub-circuits containing 1 gate and replacing them with sub-circuits containing 2 gates.

		Rule	
Label	**Condition**	$l_i \rightarrow r_i$	**Reorient?**
		⋮	
M_2		`0 = AND(-1, -1)` $\rightarrow$ `0 = NOR(-1, -1)` `1 = NAND(0, 0)`	y
M_3		`0 = AND(-1, -1)` $\rightarrow$ `0 = AND(-1, -1)` `1 = AND(-1, 0)`	y
M_4		`0 = AND(-1, -1)` $\rightarrow$ `0 = NAND(-1, -1)` `1 = NOR(0, 0)`	y
		⋮	

Table 4.4: The number of 1 gate sub-circuits which can be replaced by functionally equivalent 2 gate sub-circuits.

	Partic. Circuits	**Total Circuits**	**Percentage**
1 Input - 1 Output	6	6	100.00%
2 Inputs - 1 Output	6	6	100.00%
Total	12	12	100.00%

transformation rules displayed as rewrite rules. Appendix A contains the full list of all 176 rewrite rules.

The left-hand side of these reduction relations include the 12 sub-circuits as shown in Table 4.1. Table 4.4 provides interesting results on these left-hand sides. As shown, all 12 sub-circuits participate in reduction relations. That is to say that all 12 sub-circuits containing only 1 gate can be replaced by functionally equivalent 2 gate sub-circuits. Figure 4.3 provides a plot of this data.

Table 4.5 provides data on the frequency of the sub-circuits which participate as left-hand sides in the reduction relations. The *Min* and *Max* columns shows that each sub-circuit participates in a minimum of 12 and a maximum of 20 reduction relations. Table 4.5 also provides the means, standard deviations, and variances for the frequency of sub-circuits participating in the reduction relations.

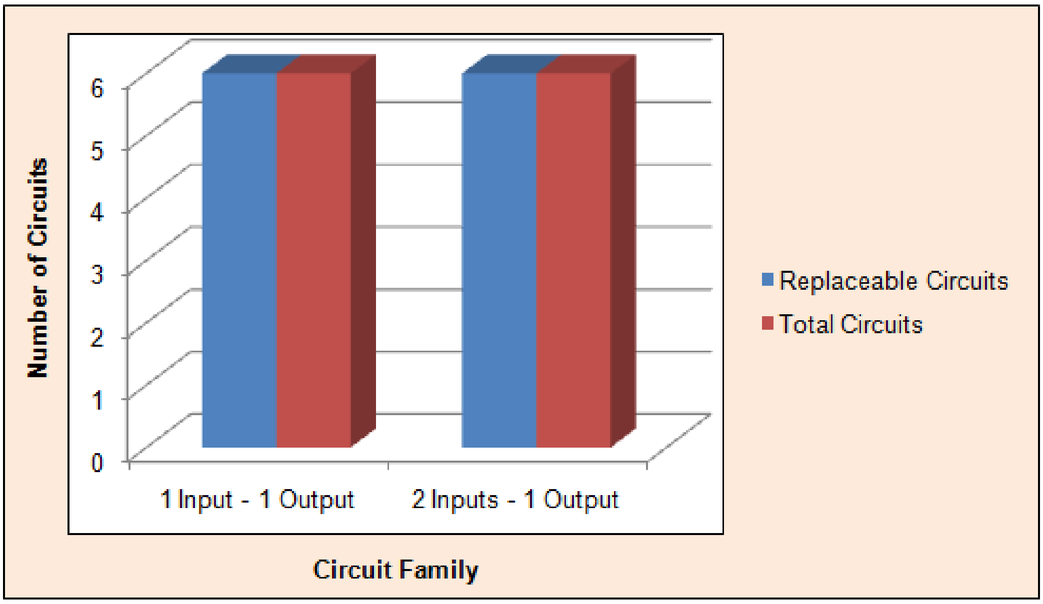

Figure 4.3: Replaceable Circuits (1 Gate Selection and 2 Gate Replacement)

Table 4.5: Circuit Selection Statistics (left-hand sides of the reduction relations).

	Mean	Std. Dev.	Variance	Min	Max
1 Input - 1 Output	17.3333	1.8856	3.5556	16	20
1 Input - 2 Outputs	12.0000	0.0000	0.0000	12	12
Total	14.6667	2.9814	8.8889	12	20

Table 4.6: The number of 2 gate sub-circuits which can replace functionally equivalent 1 gate sub-circuits.

	Partic. Circuits	Total Circuits	Percentage
1 Input - 1 Output	72	72	100.00%
2 Inputs - 1 Output	72	324	22.22%
Total	144	396	36.36%

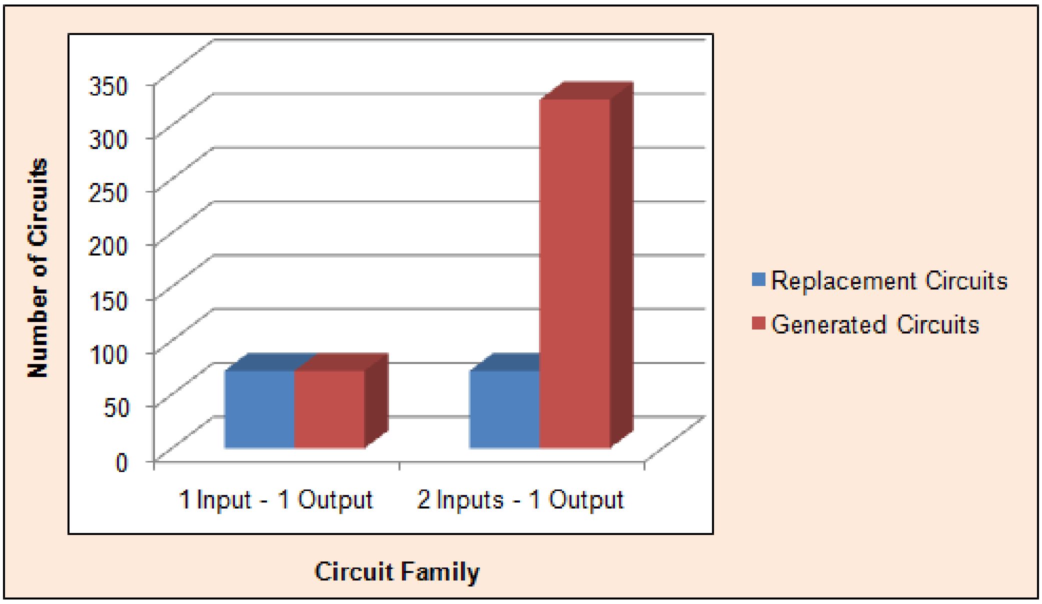

Figure 4.4: Replacement Circuits (1 Gate Selection and 2 Gate Replacement)

The right-hand side of these reduction relations include the 144 of the 396 sub-circuits shown in Table 4.1. Table 4.6 provides interesting results on the right-hand sides of the reduction rules. As shown, only 144 of the 396 sub-circuits containing 2 gates can be used for replacements of functionally equivalent 1 gate sub-circuits. Figure 4.4 provides a plot of this data.

Table 4.7 provides data on the frequency of the sub-circuits which participate in as left-hand sides in the reduction relations. The *Min* and *Max* columns shows that each sub-circuit participates in a minimum of 1 and a maximum of 2 reduction relations. Table 4.7 also provides the means, standard deviations, and variances for the frequency of sub-circuits participating in the reduction relations.

Table 4.7: Circuit Replacement Statistics (left-hand sides of the reduction relations).

	Mean	Std. Dev.	Variance	Min	Max
1 Input - 1 Output	1.4444	0.4969	0.2469	1	2
1 Input - 2 Outputs	1.0000	0.0000	0.0000	1	1
Total	1.2222	0.4157	0.1728	1	2

Table 4.8: All possible unique terms found within the sub-circuit selection and replacement algorithm.

Term
`z = AND(x, x)`
`z = AND(x, y)`
`z = OR(x, x)`
`z = OR(x, y)`
`z = NAND(x, x)`
`z = NAND(x, y)`
`z = NOR(x, x)`
`z = NOR(x, y)`
`z = NXOR(x, x)`
`z = NXOR(x, y)`
`z = XOR(x, x)`
`z = XOR(x, y)`

The next step required for the CTAS to determine if a malware detector can perfectly normalize the transformation rules of a sub-circuit selection and replacement algorithm which selects 1 gate and replaces it with 2 functionally equivalent gates is to determine if there exist any critical overlaps within the list of rewrite rules. Critical overlaps occur when the prefix of one rewrite rules matches the suffix of another rewrite rule. In a sub-circuit selection and replacement algorithm with a basis $\Omega = \{\texttt{AND}, \texttt{OR}, \texttt{NAND}, \texttt{NOR}, \texttt{NXOR}, \texttt{XOR}\}$ there are 12 unique, single gate terms shown in Table 4.8.

As shown in Table 4.9, each unique term of the sub-circuit selection and replacement algorithm participates in rewrite rules as both prefixes and suffixes.

The CTAS examines all possible rewrite rules for critical overlaps between prefixes and suffixes and determines that $2,558$ critical overlaps exist as shown in

Table 4.9: The frequency of terms in reduction relations of 1 to 2 gate transformations.

Term	**Prefix**	**Suffix**
`z = AND(x, x)`	32	16
`z = AND(x, y)`	8	17
`z = OR(x, x)`	32	16
`z = OR(x, y)`	8	17
`z = NAND(x, x)`	20	16
`z = NAND(x, y)`	4	17
`z = NOR(x, x)`	20	16
`z = NOR(x, y)`	4	17
`z = NXOR(x, x)`	16	6
`z = NXOR(x, y)`	8	16
`z = XOR(x, x)`	16	6
`z = XOR(x, y)`	8	16
Subtotals	176	176

Table 4.10: The frequency of terms in reduction relations of 1 to 2 gate transformations.

	Critical Overlaps
1 Gate Selection with 2 Gate Replacement	$2,558$

Table 4.10. Table 4.11 provides an example of one of the critical overlaps between rewrite rules M_2 and M_{21} at the term: `z = NAND (x, y)`.

As critical overlaps exist within this rule set it is by definition non-convergent and a perfect normalizer for a sub-circuit selection and replacement algorithm based on this rule set does not yet exist. However, if a completion algorithm such as the Knuth-Bendix completion procedure, the most widely used completion procedure in term rewriting literature, can complete the rule set then the rule set is convergent and an attacker can create a perfect normalizer for this sub-circuit selection and replacement algorithm.

In order to complete the rule set, the KB completion procedure iterates through each critical overlap and adds a new rule, using existing terms, to resolve the overlap. If the algorithm terminates without an error it completes the rule set and the rule set

Table 4.11: An example of a critical overlap within the reduction relations. The overlap exists between the suffix of M_2 and the prefix of M_{21}.

		Rule			
Label	**Condition**	l_i	$\rightarrow$	r_i	**Reorient?**
		⋮			
M_2		`0 = AND(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NAND(0, 0)`	y
		⋮			
M_{21}		`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = AND(0, 0)`	y
		⋮			

is convergent. However, completion algorithms are not gauranteed to terminate and may fall into a cycle of adding rules.

When the KB completion algorithm adds rules to the rule set containing the prefixes and suffixes displayed in Table 4.9, it immediately falls into a cycle. This is due to the fact that in order to resolve critical overlaps, the KB completion procedure adds rules using only prexisting terms. If every unique term of the TRS is used as a prefix and suffix at least twice, then for every rule the KB completion procedure adds, it will resolve one critical overlap while always creating at least two more. The new prefix will conflict with another suffix and the new suffix will conflict with another prefix.

As every unique term in this TRS initially participates in at least two overlaps as both a prefix and a suffix (e.g., every term $z = NAND(x, y)$ as a suffix participates in a critical overlap with each of the four equivalent prefixes as shown in Table 4.9), the KB procedure fails to terminate and the resulting rule set is non-convergent.

However, it is not possible to prove that a malware detector cannot perfectly normalize circuits obfuscated by this algorithm because even if one completion procedure, such as the Knuth-Bendix completion procedure, fails to terminate it may be the case that another completion procedure exists which can terminate the rule set. In the case of the rewrite rules based upon the transformation rules of the sub-circuit selection and replacement algorithm selecting sub-circuits containing one gate and replacing them with sub-circuits containing two gates, Knuth Bendix enters into a cycle because every term in the TRS is used more than once as both a prefix and a suffix. Therefore every rule which was added to the rule set conflicted with another rule, producing a cycle. However, it may be possible to construct a completion procedure which introduces completely new terms into the TRS which do not conflict with the current rule set. If it is possible to create such a completion procedure, then it would be possible to normalize the rule set.

4.3.2 2 Gate Selection with 3 Gate Replacement. This section displays the outputs generated by the CTAS when the number of selected gates $N_{SG} = 2$ and the number of returned gates $N_{RG} = 3$.

In this second experiment the CTAS again enumerates two lists. The first list L_S is the list of all possible 2 gate sub-circuits which can be selected within any circuit. The second list L_R is the list of all possible 3 gate sub-circuits which can be used for replacements. As shown in Table 4.12, the CTAS enumerates the list L_S of all possible sub-circuits, which can be selected within any circuit, containing only 2 gates and generates a total of $1,656$ unique sub-circuits. Also shown in Table 4.12, the CTAS enumerates the list L_R of all possible sub-circuits containing 3 gates and generates $634,824$ possible replacement sub-circuits. Figure 4.5 provides a plot of this data in in a linear scale while Figure 4.6 provides a plot of this data in a logarithmic scale.

One interesting result is that the CTAS determines that there are 0 sub-circuits which can be selected in the δ_{4-1-2} circuit family. This is intuitive as any circuit which is generated using all 4 inputs and having only 2 gates will have 2 mandatory outputs. Mandatory outputs are outputs of gates which are not connected to any other inputs. As it is not possible to generate any circuit containing 4 inputs, 2 gates, and only 1 (mandatory) output, there are 0 circuits which can be selected within the

Table 4.12: The count of all sub-circuits which are able to be used by a sub-circuit selection and replacement algorithm selecting sub-circuits containing 2 gates and replacing them with sub-circuits containing 3 gates.

	2 Gates	**3 Gates**
1 Input - 1 Output	72	1,512
1 Input - 2 Outputs	108	3,240
2 Inputs - 1 Output	180	9,720
2 Inputs - 2 Outputs	432	27,216
3 Inputs - 1 Output	108	33,696
3 Inputs - 2 Outputs	540	116,640
4 Inputs - 1 Output	0	86,400
4 Inputs - 2 Outputs	216	356,400
Subtotals	1,656	634,824

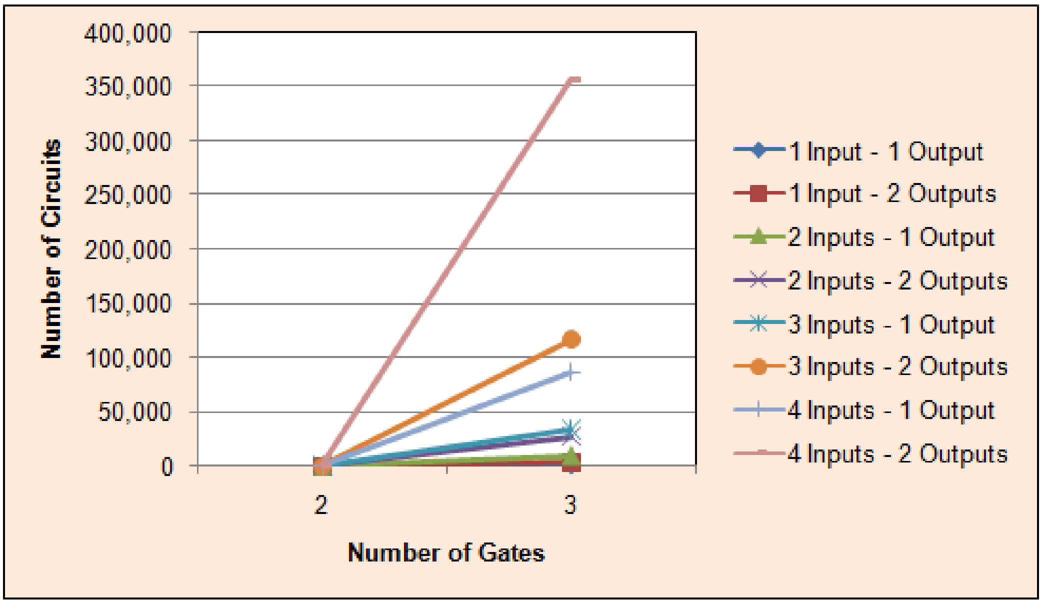

Figure 4.5: Number of Circuits (2 Gate Selection and 3 Gate Replacement)

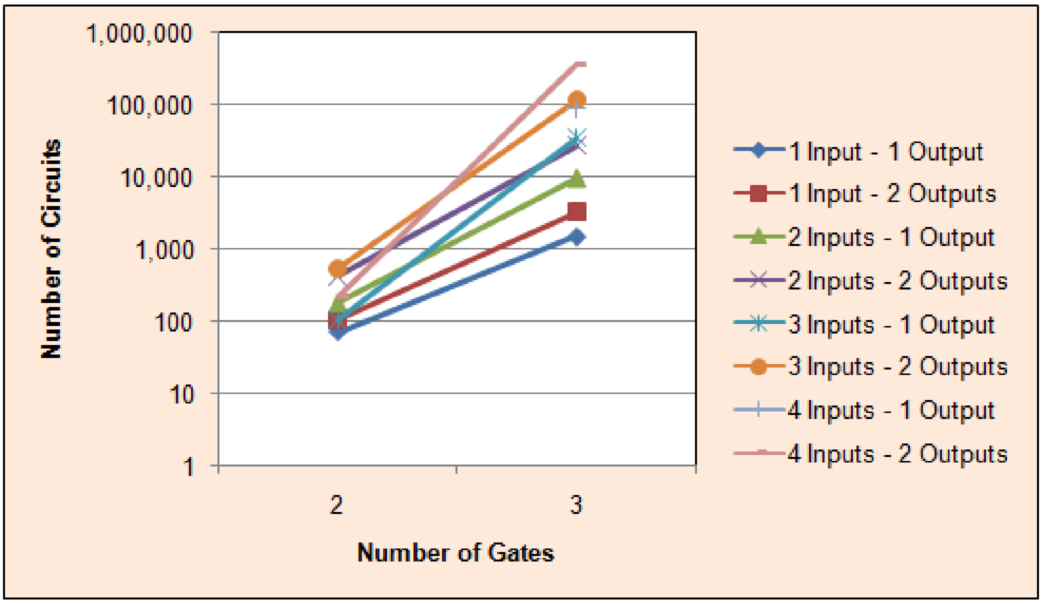

Figure 4.6: Number of Circuits (2 Gate Selection and 3 Gate Replacement) (Logarithmic Scale)

δ_{4-1-2} family. Therefore, analysis throughout this chapter will relect that the δ_{4-1-2} circuit family does not participate in any rewrite rules.

To accomplish the next step of the primary research goal, the research group determines all possible transformation rules for a sub-circuit selection and replacement algorithm selecting a sub-circuit containing 2 gates and replacing it with a sub-circuit containing 3 gates. As shown in Table 4.13, the CTAS enumerates all possible transformation rules from sub-circuits containing 2 gates to sub-circuits containing 3 gates and generates a total of 374,532 rules with a majority coming from the circuits containing 2 inputs. Figure 4.7 provides a plot of this data.

All 374,532 rules generated by the CTAS can be represented as rewrite rules in term rewriting theory. Table 4.14 provides a subset of the transformation rules displayed as rewrite rules.

The left-hand side of these reduction relations include all possible 1,656 2 gate sub-circuits as shown in autoreftab:numCircuits1to2. Table 4.15 provides interesting results on these left-hand sides. As shown, all 1,656 sub-circuits participate in reduc-

Table 4.13: The count of all transformation rules possible in a sub-circuit selection and replacement algorithm selecting sub-circuits containing 2 gates and replacing them with sub-circuits containing 3 gates.

	1 Gate to 2 Gates
1 Input - 1 Output	27,744
1 Input - 2 Outputs	37,188
2 Inputs - 1 Output	112,656
2 Inputs - 2 Outputs	119,892
3 Inputs - 1 Output	8,016
3 Inputs - 2 Outputs	58,668
4 Inputs - 1 Output	0
4 Inputs - 2 Outputs	10,368
Total	374,532

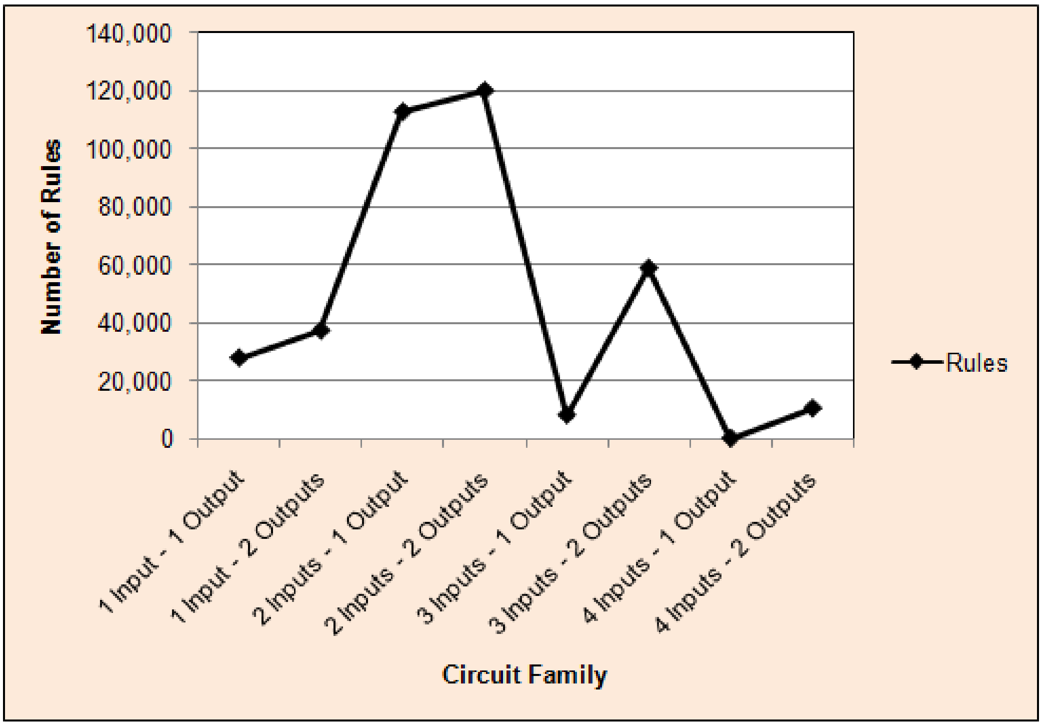

Figure 4.7: Number of Transformation Rules (2 Gate Selection and 3 Gate Replacement)

Table 4.14: A subset of the reduction relations used by a sub-circuit selection and replacement algorithm selecting sub-circuits containing 2 gates and replacing them with sub-circuits containing 3 gates.

		Rule			
Label	**Condition**	l_i	$\rightarrow$	r_i	**Reorient?**
			$\vdots$		

$M_{233,157}$		`0 = XOR(-2, -1)`
		`1 = NOR(-2, -2)`
			$\rightarrow$	`0 = NXOR(-1, -1)`
				`1 = XOR(0, -2)`
				`2 = XOR(1, -1)`
					y

$M_{233,158}$		`0 = XOR(-2, -1)`
		`1 = NOR(-2, -2)`
			$\rightarrow$	`0 = AND(-2, -1)`
				`1 = XOR(-2, -1)`
				`2 = NOR(0, -2)`
					y

$M_{233,159}$		`0 = XOR(-2, -1)`
		`1 = NOR(-2, -2)`
			$\rightarrow$	`0 = OR(-1, -1)`
				`1 = XOR(-2, -1)`
				`2 = NXOR(1, 0)`
					y

$M_{233,160}$		`0 = XOR(-2, -1)`
		`1 = NOR(-2, -2)`
			$\rightarrow$	`0 = OR(-1, -1)`
				`1 = NOR(-2, -2)`
				`2 = XOR(0, -2)`
					y

			$\vdots$		

tion relations. That is to say that all $1,656$ sub-circuits containing only 2 gates can be replaced by functionally equivalent 3 gate sub-circuits. Figure 4.8 provides a plot of this data.

Table 4.16 provides data on the frequency of the sub-circuits which participate in as left-hand sides in the reduction relations. The *Min* and *Max* columns shows that each sub-circuit participates in a minimum of 20 and a maximum of $1,756$ reduction relations (excluding the sub-circuits from the δ_{4-1} circuit family). Table 4.16 also provides the means, standard deviations, and variances for the frequency of sub-circuits participating in the reduction relations.

Table 4.17 provides interesting results on these right-hand sides. As shown, only $73,696$ sub-circuits participate in reduction relations. That is to say that only $73,696$ of the $634,824$ sub-circuits containing 3 gates can replace the functionally equivalent 2 gate sub-circuits. Figure 4.9 provides a plot of this data.

Table 4.15: The number of 2 gate sub-circuits which can be replaced by functionally equivalent 3 gate sub-circuits.

	Partic. Circuits	Total Circuits	Percentage
1 Input - 1 Output	72	72	100.0000%
1 Input - 2 Outputs	108	108	100.0000%
2 Inputs - 1 Output	180	180	100.0000%
2 Inputs - 2 Outputs	432	432	100.0000%
3 Inputs - 1 Output	108	108	100.0000%
3 Inputs - 2 Outputs	540	540	100.0000%
4 Inputs - 1 Output	0	0	100.0000%
4 Inputs - 2 Outputs	216	216	100.0000%
Total	1,656	1,656	100.0000%

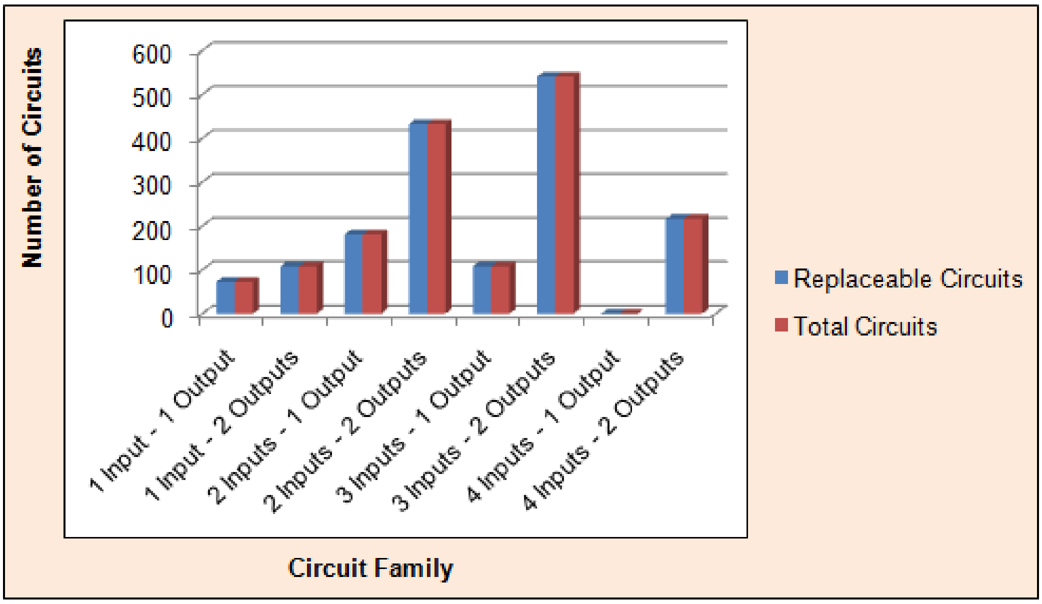

Figure 4.8: Circuits which can be Selected (2 Gate Selection and 3 Gate Replacement)

Table 4.16: Circuit Selection Statistic (left-hand sides of the reduction relations).

	Mean	**Std. Dev.**	**Variance**	**Min**	**Max**
1 Input - 1 Output	385.3333	65.5913	4,302.2222	312	444
1 Input - 2 Outputs	344.3333	87.8583	7,719.0741	176	476
2 Inputs - 1 Output	625.8667	503.3623	253,373.5822	256	1,756
2 Inputs - 2 Outputs	277.5278	279.9564	78,375.6103	61	1,668
3 Inputs - 1 Output	74.2222	42.6296	1,817.2840	44	156
3 Inputs - 2 Outputs	108.6444	114.1173	13,022.7625	20	410
4 Inputs - 1 Output	0.0000	0.0000	0.0000	0	0
4 Inputs - 2 Outputs	48.0000	0.0000	0.0000	48	48
Total	226.1667	288.5435	83,257.3466	20	1,756

Table 4.17: The number of 2 gate sub-circuits which can be replaced by functionally equivalent 3 gate sub-circuits.

	Partic. Circuits	**Total Circuits**	**Percentage**
1 Input - 1 Output	1,512	1,512	100.00%
1 Input - 2 Outputs	3240	3,240	100.00%
2 Inputs - 1 Output	9,720	9,720	100.00%
2 Inputs - 2 Outputs	22,468	27,216	82.55%
3 Inputs - 1 Output	4,752	33,696	14.10%
3 Inputs - 2 Outputs	26,820	116,640	22.99%
4 Inputs - 1 Output	0	86,400	0.00%
4 Inputs - 2 Outputs	5,184	356,400	1.45%
Total	73,696	634,824	11.61%

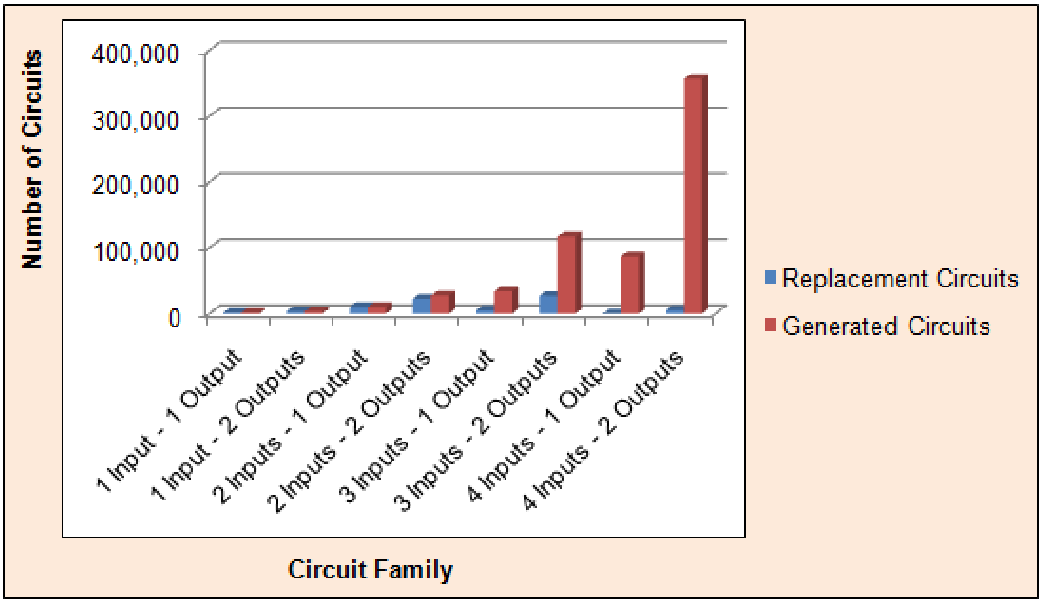

Figure 4.9: Replacement Circuits (2 Gate Selection and 3 Gate Replacement)

Table 4.16 provides data on the frequency of the sub-circuits which participate in as left-hand sides in the reduction relations. The *Min* and *Max* columns shows that each sub-circuit participates in a minimum of 20 and a maximum of $1,756$ reduction relations. Table 4.18 also provides the means, standard deviations, and variances for the frequency of sub-circuits participating in the reduction relations.

The next step required for the CTAS to determine if a malware detector can perfectly normalize the transformation rules of a sub-circuit selection and replacement algorithm which selects 1 gate and replaces it with 2 functionally equivalent gates is to determine if there exist any critical overlaps within the list of rewrite rules. As shown in Table 4.19, each unique term of the sub-circuit selection and replacement algorithm participates in rewrite rules as both prefixes and suffixes.

The CTAS examines all possible rewrite rules for critical overlaps between prefixes and suffixes and determines that $10,007,353,112$ critical overlaps exist as shown in Table 4.20.

Table 4.21 provides an example of one of the critical overlaps between rewrite rules $M_{27,788}$ and $M_{233,160}$ at the term: $z = OR(x, x)$.

Table 4.18: Circuit Replacement Statistic (right-hand sides of the reduction relations).

	Mean	Std. Dev.	Variance	Min	Max
1 Input - 1 Output	18.3492	1.9693	3.8781	16	20
1 Input - 2 Outputs	11.4778	4.2406	17.9828	5	20
2 Inputs - 1 Output	11.5901	2.4098	5.8073	8	14
2 Inputs - 2 Outputs	5.3361	1.9900	3.9600	1	8
3 Inputs - 1 Output	1.6869	1.2999	1.6898	1	6
3 Inputs - 2 Outputs	2.1875	0.8243	0.6794	1	4
4 Inputs - 1 Output	0.0000	0.0000	0.0000	0	0
4 Inputs - 2 Outputs	2.0000	0.0000	0.0000	2	2
Total	5.0821	4.3216	18.6760	1	20

Table 4.19: The frequency of terms in reduction relations of 2 to 3 gate transformations.

Term	Prefix	Suffix
`z = AND(x, x)`	48,266	17,888
`z = AND(x, y)`	19,528	42,382
`z = OR(x, x)`	48,266	17,888
`z = OR(x, y)`	19,528	42,382
`z = NAND(x, x)`	40,518	18,960
`z = NAND(x, y)`	16,262	44,024
`z = NOR(x, x)`	40,518	18,960
`z = NOR(x, y)`	16,262	44,024
`z = NXOR(x, x)`	43,418	20,298
`z = NXOR(x, y)`	19,274	43,714
`z = XOR(x, x)`	43,418	20,298
`z = XOR(x, y)`	19,274	43,714
Subtotals	374,532	374,532

Table 4.20: The frequency of terms in reduction relations of 2 to 3 gate transformations.

	Critical Overlaps
2 Gate Selection with 3 Gate Replacement	10,007,353,112

Table 4.21: An example of a critical overlap within the reduction relations. The overlap exists between the suffix of $M_{27,788}$ and the prefix of $M_{233,160}$.

		Rule				
Label	**Condition**	l_i	$\rightarrow$	r_i	**Reorient?**
			$\vdots$		

$M_{27,788}$		`0 = NOR(-1, -1)`
		`1 = AND(-1, -1)`
			$\rightarrow$	`0 = AND(-1, -1)`
				`1 = NOR(0, -1)`
				`2 = OR(-1, -1)`
					y

			$\vdots$		

$M_{233,160}$		`0 = XOR(-2, -1)`
		`1 = NOR(-2, -2)`
			$\rightarrow$	`0 = OR(-1, -1)`
				`1 = NOR(-2, -2)`
				`2 = XOR(0, -2)`
					y

			$\vdots$		

Table 4.22: An example transformation of C_1 to C_T by applying the reduction relation $M_{27,788}$.

Original Circuit C_1	**Rule**	**Transformed Circuit C_T**

`2 = NOR(0, 0)`
`3 = AND(0, 0)`
`4 = NOR(1, 1)`
`5 = XOR(2, 1)`
	$M_{27,788} \rightarrow$	`2 = AND(0, 0)`
		`3 = NOR(1, 1)`
		`4 = OR(0, 0)`
		`5 = NOR(1, 1)`
		`6 = XOR(2, 1)`

Table 4.22 provides an example of a circuit C_1 which cannot be perfectly normalized because of the critical overlap between the reduction relations $M_{27,788}$ and $M_{233,160}$. The circuit is transformed to the circuit C_T by the rule $M_{27,788}$.

Table 4.23 provides one normalization of the circuit based on the reversal of rule $M_{27,788}$, known as $N_{27,788}$. This reversal results in the original circuit C_1.

Table 4.24 provides a second (incorrect) normalization of the circuit based on the reversed rule $N_{233,160}$ which results in circuit C_2.

If there existed no other rules which could reduce C_1 and C_2, then this critical overlap would prevent a malware detector from perfectly reducing C_T to one normal form.

Table 4.23: An example of the correct normalization of C_T into C_1.

Transformed Circuit C_T	Rule	Normalized Circuit C_2
2 = AND(0, 0)	$N_{27,788} \rightarrow$	2 = NOR(0, 0)
3 = NOR(1, 1)		3 = AND(0, 0)
4 = OR(0, 0)		4 = NOR(1, 1)
5 = NOR(1, 1)		5 = XOR(2, 1)
6 = XOR(2, 1)		

Table 4.24: An example of an incorrect normalization of C_T into C_2.

Transformed Circuit C_T	Rule	Normalized Circuit C_2
2 = AND(0, 0)	$N_{233,160} \rightarrow$	2 = AND(0, 0)
3 = NOR(1, 1)		3 = NOR(2, 0)
4 = OR(0, 0)		4 = XOR(1, 0)
5 = NOR(1, 1)		5 = NOR(1, 1)
6 = XOR(2, 1)		

Once again, as critical overlaps exist within this rule set it is by definition non-convergent and a perfect normalizer for a sub-circuit selection and replacement algorithm based on this rule set does not yet exist. However, if a completion algorithm such as the Knuth-Bendix completion procedure can complete the rule set then the rule set is convergent and an attacker can create a perfect normalizer for this sub-circuit selection and replacement algorithm.

In order to complete the rule set, the KB completion procedure iterates through each critical overlap and adds a new rule, using existing terms, to resolve the overlap. If the algorithm terminates without an error it completes the rule set and the rule set is convergent. However, completion algorithms are not gauranteed to terminate and may fall into a cycle of adding rules.

When the KB completion algorithm adds rules to the rule set containing the prefixes and suffixes displayed in Table 4.19, it immediately falls into a cycle. This is due to the fact that in order to resolve critical overlaps, the KB completion procedure adds rules using only prexisting terms. If every unique term of the TRS is used as a prefix and suffix at least twice, then for every rule the KB completion procedure adds, it will resolve one critical overlap while always creating at least two more. The new prefix will conflict with another suffix and the new suffix will conflict with another prefix.

As every unique term in this TRS initially participates in at least two overlaps as both a prefix and a suffix (e.g., every term $z = NAND(x, y)$ as a suffix participates in a critical overlap with each of the four equivalent prefixes as shown in Table 4.19), the KB procedure fails to terminate and the resulting rule set is non-convergent.

However, once again, this research does not prove that a malware detector cannot perfectly normalize circuits obfuscated by this algorithm. Even though one completion procedure, the Knuth-Bendix completion procedure, fails to terminate it may be the case that another completion procedure exists which can terminate the rule set. In the case of the rewrite rules based upon the transformation rules of the sub-circuit selection and replacement algorithm selecting sub-circuits containing one gate and replacing them with sub-circuits containing two gates, Knuth-Bendix enters into a cycle because every term in the TRS is used more than once as both a prefix and a suffix. Therefore every rule which was added to the rule set conflicted with another rule, producing a cycle. However, it may be possible to construct a completion procedure which introduces completely new terms into the TRS which do not conflict with the current rule set. If it is possible to create such a completion procedure, then it would be possible to normalize the rule set.

4.4 Summary

The primary goal of this research effort is to determine if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm if the transformation algorithm is previously known. The results of this chapter have accomplished this goal by determining that it is not possible to prove that a malware detector cannot perfectly normalize the circuits transformed by a sub-circuit selection and replacement algorithm. While the Knuth-Bendix completion procedure is not able to complete the rule sets generated by the sub-circuit selection and replacement algorithm, there may exist another completion procedures which would terminate and complete the rule sets.

The secondary goal of this research effort is to determine the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and determine their effectiveness in software obfuscation. While the transforming rules of the sub-circuit selection and replacement algorithm contain critical overlaps, this does not guarantee that a completion procedure cannot complete the rule set. However, the strength of the rule set is that all possible terms are both prefixes and suffixes of rules which causes the KB procedure to cycle preventing the convergence. In order to prevent a malware detector from normalizing an obfuscated program, a software obfuscator based on command substitution must contain a rule set which cannot be completed by a completion procedure.

While it is not possible to prove that no completion procedures exist which can complete these rule sets, it may be possible to increase the cost of performing this analysis to an acceptable amount. For instance, before the completion procedure begins, an attacker must be able to first enumerate the entire rule set. If the cost of enumerating the rule set is too high (whether it would take a certain amount of time or space), this may be an effective way to prevent the attacker from normalizing the rule set for a certain amount of time. One possible way to accomplish this goal is to choose replacement sub-circuits from a random subset of a large circuit family δ_{i-o-g}. If the obfuscator is able to select replacements uniformly from the set of all replacements in a circuit family, without having to enumerate the entire family, it may be possible to utilize replacement families which will take the attacker an acceptable amount of time or space to fully enumerate. The key is that in order for an attacker to model the obfuscator as a TRS and attempt to complete the rule set, the attacker must fully enumerate all circuit families used in a sub-circuit selection and replacement algorithm in order to create the rule set.

V. Conclusions and Recommendations

5.1 Chapter Overview

The primary purpose of this chapter is to provide conclusions based on the results given in Chapter IV. This chapter will also highlight the significance of this research effort and provide recommendations for future research in the realms of software obfuscation and malware detection.

5.2 Significance of Research

There are two significant results of this research effort. This research establishes that while it is not possible to prove that a malware detector based upon the mechanisms of term rewriting thoery cannot perfectly normalize a circuit obfuscated by a sub-circuit selection and replacement algorithm, it may be possible to create a rule set which drives the runtime or storage cost of a malware detector to a high cost, preventing the attacker from obtaining a solution for an acceptable amount of time.

While Chess and White [3] suspect that perfect detection of all metamorphic malware is impossible, Walenstein *et al.* [14] claim that restricted versions of the normalization problem are solvable. Specifically, they claim that perfect normalization may be possible when an accurate model of the metamorphic engine of a malware is known. As a sub-circuit selection and replacement algorithm can be modeled as a metamorphic engine, then it is an interesting result to determine if a malware detector can perfectly normalize circuits obfuscated by this algorithm. If a malware detector can perfectly normalize a circuit, then the malware detector can reduce the original circuit as well as all all possible obfuscations of the circuit, based upon the transformation rules of the sub-circuit selection and replacement algorithm, to the same normal form.

This research effort has determined that there exist critical overlaps within the transformation rules of a sub-circuit selection and replacement algorithm which prevent a malware detector based on the mechanisms of term rewriting theory from perfectly normalizing obfuscated circuits. This is a significant result because even if a malware detector has an accurate model of the sub-circuit selection and replacement algorithm, this research effort shows perfect normalization of circuits obfuscated by this algorithm is not possible using an existing completion procedure.

However, it is not possible to prove that a malware detector cannot perfectly normalize circuits obfuscated by this algorithm because even if one completion procedure, such as the Knuth-Bendix completion procedure, fails to terminate, it may be the case that another completion procedure exists which can terminate the rule set. In the case of the rewrite rules based upon the transformation rules of the sub-circuit selection and replacement algorithm, Knuth Bendix enters into a cycle because every term in the TRS is used more than once as both a prefix and a suffix. Therefore every rule which was added to the rule set conflicted with another rule, producing a cycle. However, it may be possible to construct a completion procedure which introduces completely new terms into the TRS which do not conflict with rules. If it is possible to create such a completion procedure, then it would be possible to normalize the rule set.

This research provides significant results for the field of software obfuscation. Firstly, in order for a software obfuscator based upon command substitution to prevent perfect normalization of obfuscated programs, it must contain transformation rules which prevent known completion procedures, such as the most widely used Knuth-Bendix, from converging the rule set. This can be accomplished by inserting transformation rules which cause completion procedures such as Knuth-Bendix to cycle, thereby forcing the malware detector to use an approximation to the normalization problem such as using an incomplete rule set.

Secondly, while it is not possible to prove that no completion procedures exist which can complete a rule set, it may be possible to create a rule set which would be too costly for an attacker to analyze with a TRS. Before a completion procedure is run on a rule set, the malware detector must be able to enumerate all possible transformation rules that a metamorphic engine can use. One strength of a sub-circuit selection and replacement algorithm is that it is able to generate rules, rather than simply using stored rules. If a sub-circuit selection and replacement algorithm can dynamically create replacement circuits, without enumerating the entire i-o-g family, but rather only a random subset of the family, the sub-circuit selection and replacement algorithm may be able to use replacement circuits which exist in familys that are far too costly to generate exhaustively. In order for a malware detector based upon term rewriting theory to perfectly normalize original and obfuscated circuits, it must be able to generate all possible rules before executing a completion procedure. Therefore, a sub-circuit selection and replacement algorithm which is able to dynamically create transformation rules may be able to greatly increase the cost of a malware detector's analysis to prevent the attacker from completing the rule set (if it is even possible) for an acceptable amount of time.

5.3 Recommendations for Future Research

The primary recommendation for future research would be to create selection and replacement algorithms which can select replacement circuits from δ_{i-o-g} families with a uniform distribution without enumerating all possible sub-circuits within that family. If this is possible, then it may be possible to prevent any TRS from reducing the rule set for a certain amount of time and cost to the attacker.

Future analysis of the sub-circuit selection and replacement algorithm modeled as a TRS is also possible. While this research effort inspected the rewrite rules and critical overlaps of 1 to 2 gate and 2 to 3 gate transformations, inspecting the capabilities of other transformation combinations such as 3 to 4 gates may also provide interesting results.

Another interesting research area would be examining the effects of a sub-circuit selection and replacement algorithm which contained transformation rules that reduced the size of the circuit. Transformation rules which contain selected sub-circuits that are length-lexicographically larger than their replacement sub-circuits cannot be reoriented in a malware detector. This property of a instruction-substitution algorithm might also prevent perfect normalization.

Finally, future researchers may discover more efficient methods of generating functionally equivalent replacement sub-circuits. During this effort, the research group examined the size of the circuit families, searching for previously published integer sequences. The research group found that the integer series containing the cardinalities of the circuit families containing one input and one output with a one gate basis, as enumerated in Appendix A, are isomorphic to the integer series *A000366* enumerated in the ATT Research Online Encyclopedia of Integer Series [13]. This integer series is known as the Genocchi medians divided by 2^{n-1}. Furthermore, D. E. Knuth described the Genocchi medians as "the number of Boolean functions of n variables whose ROBDD (reduced ordered binary decision diagram) contains exactly n branch nodes, one for each variable" [13]. Considering that this research effort has uncovered a relationship between the cardinalities of the generated circuit families and ROBDDs, future research may provide more efficient algorithms for generating replacement sub-circuits based on operations to ROBDDs.

5.4 Conclusions of Research

The primary goal of this research effort is to determine if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm, even when the transformation rule set (metamorphic engine) is previously known. This goal is met when the transformation rules of a sub-circuit selection and replacement algorithm are modeled as rewrite rules in term rewriting theory and it is determined if there exist critical overlaps within these rewrite rules that cannot be resolved thereby preventing a program normalizer from converging this rule set.

The secondary goal of this research effort is to determine the properties of a sub-circuit selection and replacement algorithm which prevent the rule set from converging and to determine their effectiveness in the realm of software obfuscation. This goal is met when the cause of critical overlaps within the rule set is identified and related to the obfuscating transformations of instruction-substitution algorithms.

The primary goal of this research effort was accomplished by determining that it is not possible to prove that a malware detector based upon the mechanisms of term rewriting theory cannot perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm, even when the transformation rule set is previously known. While the Knuth-Bendix completion procedure is not able to complete the rule sets generated by the sub-circuit selection and replacement algorithm, there may be other completion procedures which would terminate and complete the rule sets.

The secondary goal of this research effort is accomplished through an analysis of the critical overlaps found within the rewrite rules in Tables 6-8 and 13-14. As the rewrite rules contain properties that prevent the Knuth-Bendix completion procedure from succesfully converging the rule set, these properties can also be used in the realm of software obfuscation. Also, this research has determined that it may be possible to dynamically create transformation rules which would prevent an attacker from completing the rule set (if it was possible) with a different completion procedure for a certain acceptable amount of time. Therefore, this research effort successfully accomplishes both research goals through an analysis of the data collected through experimentation.

1 GATE BASIS (the 1-1 family is the Genocchi Medians divided by 2^{n-1}).

Table A.1: The number of sub-circuits containing 1, 2, 3, and 4 gates

	1 Gate	**2 Gates**	**3 Gates**	**4 Gates**
1 In. - 1 Out.	1	2	7	38
1 In. - 2 Out.		3	15	111
1 In. - 3 Out.			18	162
1 In. - 4 Out.				180
1 In. - 5 Out.				
2 In. - 1 Out.	3	9	45	333
2 In. - 2 Out.		18	126	1,242
2 In. - 3 Out.			180	2,160
2 In. - 4 Out.				2,700
2 In. - 5 Out.				
3 In. - 1 Out.	6	24	156	1,464
3 In. - 2 Out.		60	540	6,660
3 In. - 3 Out.			900	13,500
3 In. - 4 Out.				18,900
3 In. - 5 Out.				
4 In. - 1 Out.	10	50	400	4,550
4 In. - 2 Out.		150	1,650	24,450
4 In. - 3 Out.			3,150	56,700
4 In. - 4 Out.				88,200
4 In. - 5 Out.				
5 In. - 1 Out.	15	90	855	11,430
5 In. - 2 Out.		315	4,095	70,875

Table A.1 – continued from previous page

	1 Gate	2 Gates	3 Gates	4 Gates
5 In. - 3 Out.			$8,820$	$185,220$
5 In. - 4 Out.				$317,520$
5 In. - 5 Out.				
6 In. - 1 Out.	21	147	$1,617$	$24,843$
6 In. - 2 Out.		588	$8,820$	$174,636$
6 In. - 3 Out.			$21,168$	$508,032$
6 In. - 4 Out.				$952,560$
6 In. - 5 Out.				
7 In. - 1 Out.	28	224	$2,800$	$48,608$
7 In. - 2 Out.		$1,008$	$17,136$	$382,032$
7 In. - 3 Out.			$45,360$	$1,224,720$
7 In. - 4 Out.				$2,494,800$
7 In. - 5 Out.				
8 In. - 1 Out.	36	324	$4,536$	$87,804$
8 In. - 2 Out.		$1,620$	$30,780$	$763,020$
8 In. - 3 Out.			$89,100$	$2,673,000$
8 In. - 4 Out.				$5,880,600$
8 In. - 5 Out.				
9 In. - 1 Out.	45	450	$6,975$	$148,950$
9 In. - 2 Out.		$2,475$	$51,975$	$1,418,175$
9 In. - 3 Out.			$163,350$	$5,390,550$
9 In. - 4 Out.				$12,741,300$
9 In. - 5 Out.				
10 In. - 1 Out.	55	605	$10,285$	$240,185$
10 In. - 2 Out.		$3,630$	$83,490$	$2,486,550$
10 In. - 3 Out.			$283,140$	$10,193,040$

Table A.1 – continued from previous page

	1 Gate	2 Gates	3 Gates	4 Gates
10 In. - 4 Out.				$25,765,740$
10 In. - 5 Out.				

Table A.2: The number of sub-circuits containing 5 and 6 gates

	5 Gates	6 Gates
1 In. - 1 Out.	295	$3,098$
1 In. - 2 Out.	$1,131$	$15,123$
1 In. - 3 Out.	$1,998$	$32,022$
1 In. - 4 Out.	$2,520$	$46,080$
1 In. - 5 Out.	$2,700$	$54,000$
2 In. - 1 Out.	$3,393$	$45,369$
2 In. - 2 Out.	$16,254$	$271,458$
2 In. - 3 Out.	$34,020$	$675,540$
2 In. - 4 Out.	$48,600$	$1,101,600$
2 In. - 5 Out.	$56,700$	$1,417,500$
3 In. - 1 Out.	$18,516$	$301,704$
3 In. - 2 Out.	$106,740$	$2,145,060$
3 In. - 3 Out.	$259,200$	$6,156,000$
3 In. - 4 Out.	$415,800$	$11,264,400$
3 In. - 5 Out.	$529,200$	$15,876,000$
4 In. - 1 Out.	$68,800$	$1,323,950$
4 In. - 2 Out.	$464,250$	$10,921,650$
4 In. - 3 Out.	$1,285,200$	$35,569,800$

Table A.2 – continued from previous page

	5 Gates	6 Gates
4 In. - 4 Out.	$2,293,200$	$72,324,000$
4 In. - 5 Out.	$3,175,200$	$111,132,000$
5 In. - 1 Out.	$201,195$	$4,468,050$
5 In. - 2 Out.	$1,556,415$	$41,983,515$
5 In. - 3 Out.	$4,842,180$	$153,124,020$
5 In. - 4 Out.	$9,525,600$	$342,921,600$
5 In. - 5 Out.	$14,288,400$	$571,536,000$
6 In. - 1 Out.	$499,065$	$12,566,883$
6 In. - 2 Out.	$4,355,316$	$132,559,308$
6 In. - 3 Out.	$15,050,448$	$535,529,232$
6 In. - 4 Out.	$32,387,040$	$1,310,722,560$
6 In. - 5 Out.	$52,390,800$	$2,357,586,000$
7 In. - 1 Out.	$1,097,488$	$30,905,504$
7 In. - 2 Out.	$10,667,664$	$361,701,648$
7 In. - 3 Out.	$40,551,840$	$1,603,838,880$
7 In. - 4 Out.	$94,802,400$	$4,261,118,400$
7 In. - 5 Out.	$164,656,800$	$8,232,840,000$
8 In. - 1 Out.	$2,201,256$	$68,555,484$
8 In. - 2 Out.	$23,585,580$	$881,686,620$
8 In. - 3 Out.	$97,831,800$	$4,258,089,000$
8 In. - 4 Out.	$246,985,200$	$12,208,125,600$
8 In. - 5 Out.	$458,686,800$	$25,227,774,000$
9 In. - 1 Out.	$4,105,575$	$140,125,050$
9 In. - 2 Out.	$48,076,875$	$1,964,558,475$
9 In. - 3 Out.	$216,112,050$	$10,266,057,450$
9 In. - 4 Out.	$586,099,800$	$31,598,424,000$

Table A.2 – continued from previous page

	5 Gates	**6 Gates**
9 In. - 5 Out.	$1,159,458,300$	$69,567,498,000$
10 In. - 1 Out.	$7,219,465$	$267,981,725$
10 In. - 2 Out.	$91,733,730$	$4,068,072,030$
10 In. - 3 Out.	$444,246,660$	$22,872,332,340$
10 In. - 4 Out.	$1,288,287,000$	$75,235,960,800$
10 In. - 5 Out.	$2,705,402,700$	$175,851,175,500$

Table A.3: The number of sub-circuits containing 7 and 8 gates

	7 Gates	**8 Gates**
1 In. - 1 Out.	$42,271$	$726,734$
1 In. - 2 Out.	$256,335$	$5,364,471$
1 In. - 3 Out.	$643,518$	$15,797,862$
1 In. - 4 Out.	$1,055,520$	$29,432,880$
1 In. - 5 Out.	$1,363,500$	$42,012,000$
2 In. - 1 Out.	$769,005$	$16,093,413$
2 In. - 2 Out.	$5,620,806$	$141,116,202$
2 In. - 3 Out.	$16,441,380$	$480,124,260$
2 In. - 4 Out.	$30,488,400$	$1,007,607,600$
2 In. - 5 Out.	$43,375,500$	$1,584,481,500$
3 In. - 1 Out.	$6,133,476$	$151,845,144$
3 In. - 2 Out.	$52,659,540$	$1,547,754,660$
3 In. - 3 Out.	$176,482,800$	$5,994,356,400$
3 In. - 4 Out.	$366,357,600$	$14,037,710,400$

Table A.3 – continued from previous page

	7 Gates	**8 Gates**
3 In. - 5 Out.	571,536,000	24,195,024,000
4 In. - 1 Out.	31,441,000	900,414,950
4 In. - 2 Out.	310,618,650	10,480,182,450
4 In. - 3 Out.	1,175,542,200	45,605,359,800
4 In. - 4 Out.	2,705,976,000	118,110,384,000
4 In. - 5 Out.	4,604,040,000	221,867,100,000
5 In. - 1 Out.	121,482,495	3,954,428,190
5 In. - 2 Out.	1,358,636,895	51,800,283,675
5 In. - 3 Out.	5,736,078,180	250,476,366,420
5 In. - 4 Out.	14,517,014,400	711,619,473,600
5 In. - 5 Out.	26,790,750,000	1,448,843,760,000
6 In. - 1 Out.	385,111,041	14,050,856,379
6 In. - 2 Out.	4,811,989,140	204,699,391,596
6 In. - 3 Out.	22,434,502,608	1,089,574,515,792
6 In. - 4 Out.	61,958,312,640	3,371,715,668,160
6 In. - 5 Out.	123,380,334,000	7,402,034,178,000
7 In. - 1 Out.	1,054,199,440	42,624,538,208
7 In. - 2 Out.	14,559,579,216	683,866,611,792
7 In. - 3 Out.	74,319,003,360	3,975,100,103,520
7 In. - 4 Out.	222,516,201,600	13,314,967,142,400
7 In. - 5 Out.	475,858,152,000	31,370,413,536,000
8 In. - 1 Out.	2,576,504,376	114,373,655,964
8 In. - 2 Out.	38,983,386,780	2,004,290,035,020
8 In. - 3 Out.	216,286,864,200	12,635,838,210,600
8 In. - 4 Out.	698,050,742,400	45,562,982,889,600
8 In. - 5 Out.	1,596,230,064,000	114,715,275,246,000

Table A.3 – continued from previous page

	7 Gates	**8 Gates**
9 In. - 1 Out.	$5,753,550,375$	$278,220,779,550$
9 In. - 2 Out.	$94,658,109,975$	$5,288,297,585,175$
9 In. - 3 Out.	$567,282,370,050$	$35,947,141,911,450$
9 In. - 4 Out.	$1,963,689,156,000$	$138,864,625,614,000$
9 In. - 5 Out.	$4,782,765,487,500$	$372,186,114,300,000$
10 In. - 1 Out.	$11,936,234,365$	$624,591,267,905$
10 In. - 2 Out.	$212,175,834,090$	$12,800,169,906,150$
10 In. - 3 Out.	$1,366,107,745,860$	$93,334,292,534,340$
10 In. - 4 Out.	$5,049,569,725,200$	$384,621,175,738,800$
10 In. - 5 Out.	$13,053,568,027,500$	$1,093,591,406,407,500$

6 GATE BASIS (Same as 1 gate basis, but multiplied by 6^n where n is the number of gates):

Table A.4: The number of sub-circuits containing 1, 2, 3, and 4 gates

	1 Gate	**2 Gates**	**3 Gates**	**4 Gates**
1 In. - 1 Out.	6	72	1,512	49,248
1 In. - 2 Out.		108	3,240	143,856
1 In. - 3 Out.			3,888	209,952
1 In. - 4 Out.				233,280
1 In. - 5 Out.				
2 In. - 1 Out.	18	324	9,720	431,568
2 In. - 2 Out.		648	27,216	1,609,632
2 In. - 3 Out.			38,880	2,799,360
2 In. - 4 Out.				3,499,200
2 In. - 5 Out.				
3 In. - 1 Out.	36	864	33,696	1,897,344
3 In. - 2 Out.		2,160	116,640	8,631,360
3 In. - 3 Out.			194,400	17,496,000
3 In. - 4 Out.				24,494,400
3 In. - 5 Out.				
4 In. - 1 Out.	60	1,800	86,400	5,896,800
4 In. - 2 Out.		5,400	356,400	31,687,200
4 In. - 3 Out.			680,400	73,483,200
4 In. - 4 Out.				114,307,200
4 In. - 5 Out.				
5 In. - 1 Out.	90	3,240	184,680	14,813,280
5 In. - 2 Out.		11,340	884,520	91,854,000

Table A.4 – continued from previous page

	1 Gate	2 Gates	3 Gates	4 Gates
5 In. - 3 Out.			$1,905,120$	$240,045,120$
5 In. - 4 Out.				$411,505,920$
5 In. - 5 Out.				
6 In. - 1 Out.	126	$5,292$	$349,272$	$32,196,528$
6 In. - 2 Out.		$21,168$	$1,905,120$	$226,328,256$
6 In. - 3 Out.			$4,572,288$	$658,409,472$
6 In. - 4 Out.				$1,234,517,760$
6 In. - 5 Out.				
7 In. - 1 Out.	168	$8,064$	$604,800$	$62,995,968$
7 In. - 2 Out.		$36,288$	$3,701,376$	$495,113,472$
7 In. - 3 Out.			$9,797,760$	$1,587,237,120$
7 In. - 4 Out.				$3,233,260,800$
7 In. - 5 Out.				
8 In. - 1 Out.	216	$11,664$	$979,776$	$113,793,984$
8 In. - 2 Out.		$58,320$	$6,648,480$	$988,873,920$
8 In. - 3 Out.			$19,245,600$	$3,464,208,000$
8 In. - 4 Out.				$7,621,257,600$
8 In. - 5 Out.				
9 In. - 1 Out.	270	$16,200$	$1,506,600$	$193,039,200$
9 In. - 2 Out.		$89,100$	$11,226,600$	$1,837,954,800$
9 In. - 3 Out.			$35,283,600$	$6,986,152,800$
9 In. - 4 Out.				$16,512,724,800$
9 In. - 5 Out.				
10 In. - 1 Out.	330	$21,780$	$2,221,560$	$311,279,760$
10 In. - 2 Out.		$130,680$	$18,033,840$	$3,222,568,800$
10 In. - 3 Out.			$61,158,240$	$13,210,179,840$

Table A.4 – continued from previous page

	1 Gate	2 Gates	3 Gates	4 Gates
10 In. - 4 Out.				$33,392,399,040$
10 In. - 5 Out.				

Table A.5: The number of sub-circuits containing 5 and 6 gates

	5 Gates	6 Gates
1 In. - 1 Out.	$2,293,920$	$144,540,288$
1 In. - 2 Out.	$8,794,656$	$705,578,688$
1 In. - 3 Out.	$15,536,448$	$1,494,018,432$
1 In. - 4 Out.	$19,595,520$	$2,149,908,480$
1 In. - 5 Out.	$20,995,200$	$2,519,424,000$
2 In. - 1 Out.	$26,383,968$	$2,116,736,064$
2 In. - 2 Out.	$126,391,104$	$12,665,144,448$
2 In. - 3 Out.	$264,539,520$	$31,517,994,240$
2 In. - 4 Out.	$377,913,600$	$51,396,249,600$
2 In. - 5 Out.	$440,899,200$	$66,134,880,000$
3 In. - 1 Out.	$143,980,416$	$14,076,301,824$
3 In. - 2 Out.	$830,010,240$	$100,079,919,360$
3 In. - 3 Out.	$2,015,539,200$	$287,214,336,000$
3 In. - 4 Out.	$3,233,260,800$	$525,551,846,400$
3 In. - 5 Out.	$4,115,059,200$	$740,710,656,000$
4 In. - 1 Out.	$534,988,800$	$61,770,211,200$
4 In. - 2 Out.	$3,610,008,000$	$509,560,502,400$
4 In. - 3 Out.	$9,993,715,200$	$1,659,544,588,800$

Table A.5 – continued from previous page

	5 Gates	6 Gates
4 In. - 4 Out.	$17,831,923,200$	$3,374,348,544,000$
4 In. - 5 Out.	$24,690,355,200$	$5,184,974,592,000$
5 In. - 1 Out.	$1,564,492,320$	$208,461,340,800$
5 In. - 2 Out.	$12,102,683,040$	$1,958,782,875,840$
5 In. - 3 Out.	$37,652,791,680$	$7,144,154,277,120$
5 In. - 4 Out.	$74,071,065,600$	$15,999,350,169,600$
5 In. - 5 Out.	$111,106,598,400$	$26,665,583,616,000$
6 In. - 1 Out.	$3,880,729,440$	$586,320,493,248$
6 In. - 2 Out.	$33,866,937,216$	$6,184,687,074,048$
6 In. - 3 Out.	$117,032,283,648$	$24,985,651,848,192$
6 In. - 4 Out.	$251,841,623,040$	$61,153,071,759,360$
6 In. - 5 Out.	$407,390,860,800$	$109,995,532,416,000$
7 In. - 1 Out.	$8,534,066,688$	$1,441,927,194,624$
7 In. - 2 Out.	$82,951,755,264$	$16,875,552,089,088$
7 In. - 3 Out.	$315,331,107,840$	$74,828,706,785,280$
7 In. - 4 Out.	$737,183,462,400$	$198,806,740,070,400$
7 In. - 5 Out.	$1,280,371,276,800$	$384,111,383,040,000$
8 In. - 1 Out.	$17,116,966,656$	$3,198,524,661,504$
8 In. - 2 Out.	$183,401,470,080$	$41,135,970,942,720$
8 In. - 3 Out.	$760,740,076,800$	$198,665,400,384,000$
8 In. - 4 Out.	$1,920,556,915,200$	$569,582,307,993,600$
8 In. - 5 Out.	$3,566,748,556,800$	$1,177,027,023,744,000$
9 In. - 1 Out.	$31,924,951,200$	$6,537,674,332,800$
9 In. - 2 Out.	$373,845,780,000$	$91,658,440,209,600$
9 In. - 3 Out.	$1,680,487,300,800$	$478,973,176,387,200$
9 In. - 4 Out.	$4,557,512,044,800$	$1,474,256,070,144,000$

Table A.5 – continued from previous page

	5 Gates	**6 Gates**
9 In. - 5 Out.	$9,015,947,740,800$	$3,245,741,186,688,000$
10 In. - 1 Out.	$56,138,559,840$	$12,502,955,361,600$
10 In. - 2 Out.	$713,321,484,480$	$189,799,968,631,680$
10 In. - 3 Out.	$3,454,462,028,160$	$1,067,131,537,655,040$
10 In. - 4 Out.	$10,017,719,712,000$	$3,510,208,987,084,800$
10 In. - 5 Out.	$21,037,211,395,200$	$8,204,512,444,128,000$

Table A.6: The number of sub-circuits containing 7 and 8 gates

	7 Gates	**8 Gates**
1 In. - 1 Out.	$11,833,174,656$	$1,220,634,054,144$
1 In. - 2 Out.	$71,757,394,560$	$9,010,251,323,136$
1 In. - 3 Out.	$180,143,854,848$	$26,534,341,780,992$
1 In. - 4 Out.	$295,478,046,720$	$49,435,936,174,080$
1 In. - 5 Out.	$381,692,736,000$	$70,564,027,392,000$
2 In. - 1 Out.	$215,272,183,680$	$27,030,753,969,408$
2 In. - 2 Out.	$1,573,465,948,416$	$237,021,030,738,432$
2 In. - 3 Out.	$4,602,534,151,680$	$806,424,389,084,160$
2 In. - 4 Out.	$8,534,800,742,400$	$1,692,393,846,681,600$
2 In. - 5 Out.	$12,142,363,968,000$	$2,661,320,479,104,000$
3 In. - 1 Out.	$1,716,980,737,536$	$255,041,533,384,704$
3 In. - 2 Out.	$14,741,300,989,440$	$2,599,633,491,010,560$
3 In. - 3 Out.	$49,403,889,100,800$	$10,068,216,919,142,400$
3 In. - 4 Out.	$102,556,681,113,600$	$23,577,962,991,206,400$

Table A.6 – continued from previous page

	7 Gates	8 Gates
3 In. - 5 Out.	159,993,501,696,000	40,638,349,430,784,000
4 In. - 1 Out.	8,801,467,776,000	1,512,351,356,659,200
4 In. - 2 Out.	86,953,342,406,400	17,602,682,125,939,200
4 In. - 3 Out.	329,076,581,299,200	76,599,492,005,836,800
4 In. - 4 Out.	757,500,097,536,000	198,380,090,732,544,000
4 In. - 5 Out.	1,288,836,541,440,000	372,651,531,033,600,000
5 In. - 1 Out.	34,007,323,720,320	6,641,920,858,775,040
5 In. - 2 Out.	380,331,377,838,720	87,004,585,265,068,800
5 In. - 3 Out.	1,605,734,781,396,480	420,704,112,660,895,000
5 In. - 4 Out.	4,063,834,943,078,400	1,195,247,453,770,140,000
5 In. - 5 Out.	7,499,695,392,000,000	2,433,501,160,796,160,000
6 In. - 1 Out.	107,806,444,373,376	23,600,043,187,870,500
6 In. - 2 Out.	1,347,048,991,895,040	343,816,373,314,907,000
6 In. - 3 Out.	6,280,224,922,073,090	1,830,066,789,916,500,000
6 In. - 4 Out.	17,344,362,207,191,000	5,663,187,583,692,230,000
6 In. - 5 Out.	34,538,597,178,624,000	12,432,575,037,915,600,000
7 In. - 1 Out.	295,108,374,435,840	71,592,856,366,768,100
7 In. - 2 Out.	4,075,750,367,410,180	1,148,633,303,031,630,000
7 In. - 3 Out.	20,804,564,524,585,000	6,676,641,735,473,850,000
7 In. - 4 Out.	62,290,295,411,097,600	22,364,031,851,849,300,000
7 In. - 5 Out.	133,209,827,638,272,000	52,690,248,501,682,200,000
8 In. - 1 Out.	721,256,328,999,936	192,103,822,535,630,000
8 In. - 2 Out.	10,912,853,361,646,100	3,366,437,611,460,150,000
8 In. - 3 Out.	60,546,479,616,691,200	21,223,356,031,935,100,000
8 In. - 4 Out.	195,409,532,624,486,000	76,528,315,069,098,400,000
8 In. - 5 Out.	446,842,259,195,904,000	192,677,611,747,586,000,000

Table A.6 – continued from previous page

	7 Gates	8 Gates
9 In. - 1 Out.	$1,610,625,877,776,000$	$467,304,072,864,653,000$
9 In. - 2 Out.	$26,498,212,673,961,600$	$8,882,309,236,821,290,000$
9 In. - 3 Out.	$158,802,757,542,317,000$	$60,377,394,708,742,000,000$
9 In. - 4 Out.	$549,707,287,574,016,000$	$233,239,247,015,284,000,000$
9 In. - 5 Out.	$1,338,868,239,508,800,000$	$625,129,752,556,109,000,000$
10 In. - 1 Out.	$3,341,381,703,200,640$	$1,049,073,487,033,520,000$
10 In. - 2 Out.	$59,395,654,291,818,200$	$21,499,370,177,088,000,000$
10 In. - 3 Out.	$382,422,737,945,065,000$	$156,765,771,089,358,000,000$
10 In. - 4 Out.	$1,413,556,350,593,590,000$	$646,015,880,709,700,000,000$
10 In. - 5 Out.	$3,654,163,619,346,240,000$	$1,836,813,623,664,540,000,000$

This is the table of rewrite rules used by a sub-circuit selection and replacement algorithm selecting sub-circuits containing one gate and replacing them with sub-circuits containing two gates:

Table B.1: Circuit Transformation Rules

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_1	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NOR(0, 0)`	y
M_2	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NAND(0, 0)`	y
M_3	`0 = AND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = AND(0, -1)`	y
M_4	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NOR(0, 0)`	y
M_5	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NAND(0, 0)`	y
M_6	`0 = AND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = OR(0, -1)`	y
M_7	`0 = AND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = AND(0, 0)`	y
M_8	`0 = AND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = OR(0, 0)`	y
M_9	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = AND(0, -1)`	y

Table B.1 – continued from previous page

	Rule			Reorient?
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{10}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{11}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = AND(0, -1)`	y
M_{12}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = OR(0, -1)`	y
M_{13}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = AND(0, 0)`	y
M_{14}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = OR(0, 0)`	y
M_{15}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = XOR(0, -1)`	y
M_{16}	`0 = AND(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = OR(0, -1)`	y
M_{17}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = AND(0, 0)`	y
M_{18}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = OR(0, 0)`	y
M_{19}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NOR(0, -1)`	y
M_{20}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NAND(0, -1)`	y
M_{21}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = AND(0, 0)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{22}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = OR(0, 0)`	y
M_{23}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NOR(0, 0)`	y
M_{24}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NAND(0, 0)`	y
M_{25}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NAND(0, -1)`	y
M_{26}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = XOR(0, -1)`	y
M_{27}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NOR(0, -1)`	y
M_{28}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NAND(0, -1)`	y
M_{29}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NOR(0, 0)`	y
M_{30}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NAND(0, 0)`	y
M_{31}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NOR(0, -1)`	y
M_{32}	`0 = NOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{33}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = AND(0, 0)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{34}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = OR(0, 0)`	y
M_{35}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NOR(0, -1)`	y
M_{36}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NAND(0, -1)`	y
M_{37}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = AND(0, 0)`	y
M_{38}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = OR(0, 0)`	y
M_{39}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NOR(0, 0)`	y
M_{40}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NAND(0, 0)`	y
M_{41}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NAND(0, -1)`	y
M_{42}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = XOR(0, -1)`	y
M_{43}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NOR(0, -1)`	y
M_{44}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NAND(0, -1)`	y
M_{45}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NOR(0, 0)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{46}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NAND(0, 0)`	y
M_{47}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NOR(0, -1)`	y
M_{48}	`0 = NAND(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{49}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NAND(0, -1)`	y
M_{50}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = XOR(0, -1)`	y
M_{51}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = OR(0, -1)`	y
M_{52}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NXOR(0, 0)`	y
M_{53}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NAND(0, -1)`	y
M_{54}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = XOR(0, -1)`	y
M_{55}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = OR(0, -1)`	y
M_{56}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NXOR(0, 0)`	y
M_{57}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NXOR(0, -1)`	y

Table B.1 – continued from previous page

	Rule			Reorient?
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{58}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NXOR(0, 0)`	y
M_{59}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = OR(0, -1)`	y
M_{60}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = AND(0, 0)`	y
M_{61}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NXOR(0, 0)`	y
M_{62}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = OR(0, 0)`	y
M_{63}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{64}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NXOR(0, 0)`	y
M_{65}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NAND(0, -1)`	y
M_{66}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NOR(0, 0)`	y
M_{67}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NAND(0, 0)`	y
M_{68}	`0 = NXOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = NXOR(0, 0)`	y
M_{69}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = AND(0, -1)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{70}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NOR(0, -1)`	y
M_{71}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{72}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = XOR(0, 0)`	y
M_{73}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = AND(0, -1)`	y
M_{74}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NOR(0, -1)`	y
M_{75}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NXOR(0, -1)`	y
M_{76}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = XOR(0, 0)`	y
M_{77}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = XOR(0, -1)`	y
M_{78}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = XOR(0, 0)`	y
M_{79}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NOR(0, -1)`	y
M_{80}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NOR(0, 0)`	y
M_{81}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NAND(0, 0)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{82}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = XOR(0, 0)`	y
M_{83}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = XOR(0, -1)`	y
M_{84}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = XOR(0, 0)`	y
M_{85}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = AND(0, -1)`	y
M_{86}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = AND(0, 0)`	y
M_{87}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = XOR(0, 0)`	y
M_{88}	`0 = XOR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = OR(0, 0)`	y
M_{89}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NOR(0, 0)`	y
M_{90}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NAND(0, 0)`	y
M_{91}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = AND(0, -1)`	y
M_{92}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NOR(0, 0)`	y
M_{93}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NAND(0, 0)`	y

Table B.1 – continued from previous page

	Rule			Reorient?
Label	l_i	$\rightarrow$	r_i	Reorient?
M_{94}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = OR(0, -1)`	y
M_{95}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = AND(0, 0)`	y
M_{96}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = OR(0, 0)`	y
M_{97}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = AND(0, -1)`	y
M_{98}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = NXOR(-1, -1)` `1 = NXOR(0, -1)`	y
M_{99}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = AND(0, -1)`	y
M_{100}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = OR(0, -1)`	y
M_{101}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = AND(0, 0)`	y
M_{102}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = OR(0, 0)`	y
M_{103}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = XOR(0, -1)`	y
M_{104}	`0 = OR(-1, -1)`	$\rightarrow$	`0 = XOR(-1, -1)` `1 = OR(0, -1)`	y
M_{105}	`0 = AND(-2, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = AND(0, -2)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\to$	r_i	**Reorient?**
M_{106}	`0 = AND(-2, -1)`	$\to$	`0 = OR(-1, -1)` `1 = AND(0, -2)`	y
M_{107}	`0 = AND(-2, -1)`	$\to$	`0 = AND(-2, -1)` `1 = AND(0, -1)`	y
M_{108}	`0 = AND(-2, -1)`	$\to$	`0 = AND(-2, -1)` `1 = AND(0, -2)`	y
M_{109}	`0 = AND(-2, -1)`	$\to$	`0 = AND(-2, -1)` `1 = AND(0, 0)`	y
M_{110}	`0 = AND(-2, -1)`	$\to$	`0 = AND(-2, -1)` `1 = OR(0, 0)`	y
M_{111}	`0 = AND(-2, -1)`	$\to$	`0 = NAND(-2, -1)` `1 = NOR(0, 0)`	y
M_{112}	`0 = AND(-2, -1)`	$\to$	`0 = NAND(-2, -1)` `1 = NAND(0, 0)`	y
M_{113}	`0 = AND(-2, -1)`	$\to$	`0 = NXOR(-2, -1)` `1 = AND(0, -1)`	y
M_{114}	`0 = AND(-2, -1)`	$\to$	`0 = NXOR(-2, -1)` `1 = AND(0, -2)`	y
M_{115}	`0 = AND(-2, -1)`	$\to$	`0 = AND(-2, -2)` `1 = AND(0, -1)`	y
M_{116}	`0 = AND(-2, -1)`	$\to$	`0 = OR(-2, -2)` `1 = AND(0, -1)`	y
M_{117}	`0 = NOR(-2, -1)`	$\to$	`0 = AND(-1, -1)` `1 = NOR(0, -2)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{118}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NOR(0, -2)`	y
M_{119}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -1)` `1 = AND(0, 0)`	y
M_{120}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -1)` `1 = OR(0, 0)`	y
M_{121}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = NOR(0, -1)`	y
M_{122}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = NOR(0, -2)`	y
M_{123}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = NOR(0, -1)`	y
M_{124}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = NOR(0, -2)`	y
M_{125}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = NOR(0, 0)`	y
M_{126}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = NAND(0, 0)`	y
M_{127}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = AND(-2, -2)` `1 = NOR(0, -1)`	y
M_{128}	`0 = NOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -2)` `1 = NOR(0, -1)`	y
M_{129}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NAND(0, -2)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{130}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NAND(0, -2)`	y
M_{131}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-2, -1)` `1 = NAND(0, -1)`	y
M_{132}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-2, -1)` `1 = NAND(0, -2)`	y
M_{133}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-2, -1)` `1 = NOR(0, 0)`	y
M_{134}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-2, -1)` `1 = NAND(0, 0)`	y
M_{135}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = NAND(-2, -1)` `1 = AND(0, 0)`	y
M_{136}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = NAND(-2, -1)` `1 = OR(0, 0)`	y
M_{137}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = NAND(0, -1)`	y
M_{138}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = NAND(0, -2)`	y
M_{139}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = AND(-2, -2)` `1 = NAND(0, -1)`	y
M_{140}	`0 = NAND(-2, -1)`	$\rightarrow$	`0 = OR(-2, -2)` `1 = NAND(0, -1)`	y
M_{141}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = XOR(0, -2)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{142}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = NXOR(0, -2)`	y
M_{143}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = XOR(0, -2)`	y
M_{144}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = NXOR(0, -2)`	y
M_{145}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = NOR(0, 0)`	y
M_{146}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = NAND(0, 0)`	y
M_{147}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = AND(0, 0)`	y
M_{148}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = OR(0, 0)`	y
M_{149}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -2)` `1 = XOR(0, -1)`	y
M_{150}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = AND(-2, -2)` `1 = NXOR(0, -1)`	y
M_{151}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = NAND(-2, -2)` `1 = XOR(0, -1)`	y
M_{152}	`0 = NXOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -2)` `1 = NXOR(0, -1)`	y
M_{153}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NOR(-1, -1)` `1 = NXOR(0, -2)`	y

Table B.1 – continued from previous page

	Rule			
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{154}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = XOR(0, -2)`	y
M_{155}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NAND(-1, -1)` `1 = NXOR(0, -2)`	y
M_{156}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = XOR(0, -2)`	y
M_{157}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = AND(0, 0)`	y
M_{158}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = OR(0, 0)`	y
M_{159}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = NOR(0, 0)`	y
M_{160}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NXOR(-2, -1)` `1 = NAND(0, 0)`	y
M_{161}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -2)` `1 = NXOR(0, -1)`	y
M_{162}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = AND(-2, -2)` `1 = XOR(0, -1)`	y
M_{163}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = NAND(-2, -2)` `1 = NXOR(0, -1)`	y
M_{164}	`0 = XOR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -2)` `1 = XOR(0, -1)`	y
M_{165}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = AND(-1, -1)` `1 = OR(0, -2)`	y

Table B.1 – continued from previous page

	Rule			Reorient?
Label	l_i	$\rightarrow$	r_i	**Reorient?**
M_{166}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-1, -1)` `1 = OR(0, -2)`	y
M_{167}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -1)` `1 = NOR(0, 0)`	y
M_{168}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = NOR(-2, -1)` `1 = NAND(0, 0)`	y
M_{169}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = OR(0, -1)`	y
M_{170}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = XOR(-2, -1)` `1 = OR(0, -2)`	y
M_{171}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = OR(0, -1)`	y
M_{172}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = OR(0, -2)`	y
M_{173}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = AND(0, 0)`	y
M_{174}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -1)` `1 = OR(0, 0)`	y
M_{175}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = AND(-2, -2)` `1 = OR(0, -1)`	y
M_{176}	`0 = OR(-2, -1)`	$\rightarrow$	`0 = OR(-2, -2)` `1 = OR(0, -1)`	y

Bibliography

1. Baader, Franz and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-45520-0.
2. Barak, Boaz, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (Im)possibility of obfuscating programs". *Electronic Colloquium on Computational Complextiy*, 8(57):1–41, 2001.
3. Chess, David M. and Steve R. White. "An undetectable computer virus". *In Proceedings of Virus Bulletin Conference*. 2000.
4. Goldwasser, Shafi and Guy N. Rothblum. "On Best-Possible Obfuscation". *4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, 194–213. Springer, 21-24 February 2007. ISBN 3-540-70935-5.
5. Knuth, D.E. and P.B. Bendix. "Simple Word Problems in Universal Algebra". J. Leech (editor), *Computational Word Problems in Abstract Algebra*, 263–297. Pergamon Press, 1970.
6. Lakhotia, Arun and Moinuddin Mohammed. "Imposing Order on Program Statements to Assist Anti-Virus Scanners". *Reverse Engineering, Working Conference on*, 0:161–170, 2004. ISSN 1095-1350.
7. McDonald, J. T. and Alec Yasinsac. "Program intent protection using circuit encryption". *Proc of the 8th Intl Symposium on System and Information Security*. IEEE Computer Society, 2006.
8. McDonald, J. Todd and Moses James. *Considering Obfuscation Metrics in the Random Program Model*. Technical report, Air Force Institute of Technology, Wright Patterson AFB, OH, USA, 2008.
9. McDonald, Jeffrey T. *Enhanced Security for Mobile Agent Systems*. Ph.D. thesis, Florida State University, 2006.
10. Preda, Mila Dalla. *Code Obfuscation and Malware Detection by Abstract Interpretation*. Ph.D. thesis, Universita' di Verona, 2007.
11. Preda, Mila Dalla, Mihai Christodorescu, Somesh Jha, and Saumya Debray. "A semantics-based approach to malware detection". *ACM Trans. Program. Lang. Syst.*, 30(5):1–54, 2008. ISSN 0164-0925.
12. Rajgopal, Suresh. *Spatial Entropy - A Unified Attribute to Model Dynamic Communication in VLSI Circuits*. Technical report, Chapel Hill, NC, USA, 1992.
13. Sloane, N. J. A. "ATT Research Online Encyclopedia of Integer Series". Internet: `www.research.att.com/~njas/sequences/`, 2008.

14. Walenstein, Andrew, Rachit Mathur, Mohamed R. Chouchane, and Arun Lakhotia. "Normalizing Metamorphic Malware Using Term Rewriting". *SCAM '06: Proceedings of the Sixth IEEE International Workshop on Source Code Analysis and Manipulation*, 75–84. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2353-6.

15. Yasinsac, Alec and J. Todd McDonald. "Of unicorns and random programs". *Communications and Computer Networks*, 24–30. 2005.

Vita

Eric D. Simonaire graduated from Granite Baptist Church School in Glen Burnie, Maryland. In 2003, Eric entered undergraduate studies at Cedarville University in Cedarville, Ohio where he graduated with a Bachelor degree in Computer Science in 2007.

While an undergraduate, Eric Simonaire was selected as a recipient of the Federal Cyber Service Scholarship for Service to attend the Air Force Institute of Technology in 2007. Upon graduation, he will serve the federal government in an Information Assurance position.

Index

The index is conceptual and does not designate every occurrence of a keyword. Page numbers in bold represent concept definition or introduction.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
16-12-2008	Master's Thesis	Sep 2007–Dec 2008

4. TITLE AND SUBTITLE

Sub-circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Simonaire, Eric D.

5d. PROJECT NUMBER

08-183

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCO/ENG/09-02

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Robert Herklotz
Air Force Office of Scientific Research
801 North Randolph Street, Rm 732
Arlington VA 22203-1977
703–696–9544 (DSN: 426)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Intent protection is a model of software obfuscation which, among other criteria, prevents an adversary from understanding the program's function for use with contextual information. Relating this framework for obfuscation to malware detection, if a malware detector can perfectly normalize a program P and any obfuscation (variant) of the program $O(P)$, the program is not intent protected. The problem of intent protection on programs can also be modeled as intent protection on combinational logic circuits. If a malware detector can perfectly normalize a circuit C and any obfuscation (variant) $O(C)$ of the circuit, the circuit is not intent protected.

In this effort, the research group set the primary goal as determining if a malware detector based upon the mechanisms of term rewriting theory can perfectly normalize circuits transformed by a sub-circuit selection and replacement algorithm, even when the transformation algorithm is known. The research group set the secondary goal as relating this result on circuit transformations to the realm of software obfuscation. The transformation rules of the sub-circuit selection and replacement algorithm are identified and modeled as rewrite rules in a term rewriting system. These rewrite rules are examined for critical overlaps which cannot be resolved by a widely used completion algorithm known as Knuth-Bendix. The research group performs an analysis of the critical overlaps found within the rewrite rules and successfully relates these results to the instruction-substitution obfuscations of a software obfuscator.

15. SUBJECT TERMS

software obfuscation, metamorphic malware, malware detection, perfect normalization, term rewriting system

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
					Lt Col J. Todd McDonald
U	U	U	UU	101	19b. TELEPHONE NUMBER *(include area code)* 937–255–3636 x4639, jmcdonal@afit.edu