Understanding how to attack DES via SAT

William Jones - 635773

December 13, 2013

Contents

1 Introduction

This document includes an outline and explanation of the introduction about the basics of Sets, syntax and semantics for propositional logic, CNF and deriving a satisfying assignment which will be used to identify that a given formula can be used to break DES.

1.1 Aims of the project

The aims that were originally listed in the initial document as:

1. Research and study the Data Encryption Standard.
2. Understand the fundamental basics of satisfiability and Boolean functions.
3. To Understand and document the existing translation of DES to SAT.
4. To develop a Data Encryption Standard via Conjunctive Normal Form.
5. Test the CNF's in a SAT solver and find methods to improve aspects of the computation
6. Show improvements of the existing facilities with the use of experiments

No changes are needed to be made to the project aims as the project aims seem reasonable. However there are personal aims of the project that are not included but should still be noted, which are:

1. Install and use the software LaTeX to write the dissertation
2. Install and understand how to use git and github, to keep track of the progress of the project with the supervisor.

LaTeX was mentioned in section 3.2 of the Initial document as the program to be used to produce a clear and formatted research paper. At the end, when the paper is complete, I should have an understanding of how to write fluently within the software. This is a personal aim of the project and therefore is not included within the main aims of the project, but it is still important to note. Although it is not mentioned within the software applications in chapter 3 of the initial document, it is important to understand that the software is being used to help communicate with my supervisor to ensure that the project stays on course and updates of a document can be made by each other in real time. As said before, this is a side aim of the project and is not directly important to the success of the project, but helps achieve these aims.

2 Review of progress

Since the initial document, the SAT chapter has been completed, with an outline of what satisfiability is. Definitions and explanations have been formed to define many of the fundamental layers of logic needed to derive satisfiability. More specifically, sets, subsets and ordered sets have been defined to show what is in a set. The four basic operations have been defined and shown how sets can interact with each other. The syntax and semantics for propositional logic has been introduced to show how the language can be used to build up logical formulas consisting of variables, literals, clauses and sets of clauses. A closely linked subject Boolean logic was also introduced and an explanation of the logical functors were added. A definition of both Partial Assignment and Total Assignment were added with how they are constructed with an example of a partial assignment. It was noted in feedback from the initial document that CNF was in the terminology but was not explained. As a result CNF has now been added and explained with use of material that has been explained before such as the clauses. This helps build a concrete understanding of what a CNF is and how it is used. Within the CNF topic, it explains how to convert a normal propositional formula into a CNF using 3 different laws. A section about converting CNF to clause set logic was added to show how it takes place and what laws to use when converting. As well as this examples are given. The end of the chapter shows an example of a formula being satisfiable. The second part of progress was introducing Github as a Software application to be used within the project.

Some aims have already been met.In terms of progress the project is currently at stage 4 of the aims. However stage 4 will take the longest time as it is a big part of the project.

2.1 Improvements to the Initial document

After reviewing the initial document and looking at the feedback, it was clear that some explanations that were given lacked depth or were not explained very well which could leave the reader confused. In order to combat this, a revised version of said explanations were reworded so that they are clearer and more precise. These changes were first of all to explain and actually show how a CNF was constructed (see **3.3 Syntax and semantics for propositional logic**). Additionally it was noted that some computations of the DES that was described may have been wrong and so corrections were made. This is because the DES needs to be correct in its computation in order to understand how to break it.

The decryption notation was previously wrong as it was

$$P = E(K, C)$$

However this is technically wrong for decryption as E stranded for encryption where as it should have been

$$P = D(K, C)$$

This new notation means that decryption takes place when putting together a Cipher text and key together in the encryption algorithm which will output a Plain text. Additionally the definition of an intervertebral function has been introduced. An intervertebral function allows you to carry out the same process from getting state 1 to state 2. As a result the same process allows us to get from state 2 to state 1. In this case, the same key carry out the same function to both encrypt and decrypt a message.

Related work has been updated with other cryptanalysis techniques that can be used to break DES. Other Cryptanalysis techniques include exhaustive key search, which is other wise known as brute force. This technique attempts to try every combination possible in order to try and find the key to the coded message. However the computation time to try all possible combinations would be 2^{56}, which is roughly 72 quadrillion possible keys. Other cryptanalysis techniques include differential cryptanalysis. Differential cryptanalysis was the first method to break DES faster than the previous brute force method that the exhaustive search did. The method is based up having a device which can encrypt data with a hard-wired secret key, and we assume that we don't have the tools to "read" the key within a chip. What we then do, is to choose some blocks of data and to encrypt them with the device. The data analysis will then compute the key by analysing roughly 2^47 chosen plain text bits [1]. This reduced the probability of success as we cut out a lot of computation compared to 2^{56}.

3 Sets

A set is a collection of objects. To show that an object x is in the set X we use the membership symbol ($\in$) to show that x belongs in something. This will be written as $x \in X$. Sets can be elements of other sets which are referred to as subsets. To show that all subsets of a set X we write it as $\mathcal{P}(X)$ [2]. Two sets are equivalent if they have the same exact elements in both of the sets. It should be noted that although the two sets have the same elements, the sets themselves can mean different things. Set X is in set Y if all elements of X are an element of Y. Therefore we can say for the sets X and Y:

$$X = Y \text{ if and only if } X \subseteq Y \text{ and } Y \subseteq X.$$

A set can have nothing within inside it (no elements) which is referred as the empty set. The empty set is denoted as the symbol $\emptyset$ or $\{\}$. There are four operations that can be applied to sets that can produce new sets.

3.1 The four basic operations

The *intersection* operation takes two sets and finds the element that occur in both set X and set Y, $X \cap Y$. Therefore intersection is:

$$X \cap Y = \{x : x \in X \text{ and } x \in Y\}.$$

The *union* operation takes two sets to find an element that is in either both of the two sets or belongs in the set X or Y, $X \cup Y$. Therefore the union of sets X and Y is:

$$X \cup Y = \{x : x \in X \text{ or } x \in Y\}.$$

The *difference* operation finds an element that is in set X but not in set Y, $X \setminus Y$. Therefore the difference of sets X and Y is:

$$X \setminus Y = \{x : x \in X \text{ and } x \notin Y\}.$$

The sets we have been talking about up to this point have elements within them that are not dependant on the order they are in. For example the set $\{x, y\}$ is the same as the set $\{y, x\}$ as they both have the same elements. However we can have ordered sets were the position of a element within the set is important and makes the set unique. To define a set that is ordered we use parentheses "()" , i.e., (x, y). The (Cartesian) product of sets X, Y is

$$X \times Y = \{(x, y) : x \in X \text{ and } y \in Y\}.$$

A *relation* is a subset of the product of sets, which is written as $R \subseteq X \times Y$ [2]. A *function* relates an input to an output. Functions are denoted by a small "f" followed by its input. A simple example of this would be $f(x) = x^2$. Using the variable x for when $x = 4$ the function will therefore be, $f(x) = 16$.

3.2 SAT

SAT is the problem of propositional satisfiability for formulas that are in CNF. Many examples of a SAT are can be solved easily, but other problems can have a worst case scenario which gives a long computational time. This section is an intoduction to solving these problems by defining a logical preliminary to build a frame work.

3.3 Syntax and semantics for propositional logic

A propositional variable is the starting point of propositional logic and is a alphabetic symbol that represents a object or number which is subject to change or is not known. We will refer to a set of variables as $\boldsymbol{VA}$. Let a stand for all variables, $a \in \boldsymbol{VA}$. The symbol $\top$ will be used to show that a formula is always true and $\bot$ for when a formula is always false. The symbol $\neg$ will be used to show the negation of an object. The binary symbols $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implies) and $\equiv$ (equivalence) will be used throughout this paper and are known as *functors* (see [2]). A propositional formula is defined as a set of strings over the set Var such that $\{Form : a, \neg, \top, \bot, \wedge, \vee, \Rightarrow, \equiv \in Form\}$. A literal is a variable that is either positive or negative (denoted by the negation symbol) which will be referred to as $\boldsymbol{LIT} = \boldsymbol{VA} \cup \{\bar{v} : v \in \boldsymbol{VA}\}$, for a set of literals. A clause is a formula in the form $l_1 \vee \ldots \vee l_k$, where each l_j, $1 \leq j \leq k$ is a *literal* (see[2]). We will refer to a set of clauses as $\boldsymbol{CL} := \{\mathrm{C} \subseteq \boldsymbol{LIT}\ \mathrm{C} \cap \bar{C} = \emptyset\}$. A set of clause-set's contains $\boldsymbol{CL}$ which we will refer as $\boldsymbol{CLS} := \{F \subseteq \boldsymbol{CL}\}$.

3.4 Boolean logic

Boolean logic is referred to as two valued logic. This is because two values are used to determine if something is true of false. *True* is represented as 1 and *false* is 0. The structure of Boolean logic is:

$$\mathrm{Bool} = \langle \vee, \wedge, \neg, \Rightarrow, \equiv, \{0,1\}, 0, 1 \rangle$$

The arguments for the Bool algebra operations can be shown in truth tables. A truth table computes the logical values that are given to their corresponding value over a given operation. An example of this would be the negation of p which would be $\neg p$.

p	$\neg p$
0	1
1	0

Figure 1: negation

As seen in *Figure 1* the $\neg p$ will always have a value that is opposite p. The logical *functors* each have a unique output if called upon using a set of Bool elements.

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \equiv q$	$p \oplus q$
1	1	1	1	1	1	0
1	0	0	1	0	0	1
0	1	0	1	1	0	1
0	0	0	0	1	1	0

Figure 2: Four binary functors

The *Conjunction* functor ($\wedge$) takes two Bool elements from a set $\{0, 1\}$ and will return true if the elements of both elements are 1. If however either one of the two elements or both are 0, then the Conjunction of the two elements will return 0. The *Disjunction* functor ($\vee$) takes two Bool elements from a set $\{0, 1\}$ and will return a true value if both elements are valued as 1, or either one of the two elements are equal to 1. If however neither of the two elements are equal to 1, then the Disjunction will return 0. The *Implication* functor ($\Rightarrow$) takes two Bool from a set $\{0, 1\}$ and will return true if both elements are equal to 1, both elements are equal to 0 or the first element is 0 and the second element is 1, as this says that false implies true which is true. However the implication functor will return true if the elements assignments are for the first element 1 and the second is 0, as this implies that truth implies false which is false. *Equivalence* functor takes two Bool elements from a set $\{0, 1\}$ and will return true if the elements assignments are both either 1 or 0. However, if one of the two elements is 1 while the other is 0, then the equivalence of the two elements will evaluate to 0.

3.5 Partial Assignment and Total Assignment

A partial assignment ($\boldsymbol{PASS}$) creates a mathematical object which can be instantiated by applying an instance of it to a clause set. Within the partial assignment there will be some undefinable variables. $\boldsymbol{PASS}$ is the set of all partial assignments. A total assignment ($\boldsymbol{TASS}$) is an assignment to all literals such that $\{\varphi \in \boldsymbol{TASS} : \varphi \in Var \wedge \varphi \in \{0, 1\}\}$. The task is to define $\varphi \times F$ for partial assignment φ and clause set F. It should be noted that if a partial assignment has a mapping such that all variables have a assigned value $\varphi \in \{0, 1\}$, then $\boldsymbol{PASS} = \boldsymbol{TASS}$. The relation of a partial assignment and a $\boldsymbol{CSL}$ can be denoted as:

$$* : \boldsymbol{PASS} \times \boldsymbol{CSL} \to \boldsymbol{CSL}$$

This will give us $\top := \emptyset$, so now we can have $\top \in \boldsymbol{CSL}$. Therefore:

$$\varphi * F = \top$$

φ is a satisfying assignment for F if $\varphi* = \top$. A clause set is satisfiable if there exists a partial assignment φ which satisfies F, i.e., $\varphi* = \top$.

3.6 Example of *PASS*

$<>\in$ ***PASS*** is an example of the empty partial assignment. $< v \rightarrow \epsilon >$ where$v \in Var \wedge v \in \{0,1\}$. v is assigned to ϵ which is within the ***PASS*** where v is also found as a member in the Var and is either 0 or 1.

$$A \vee 0 \rightarrow 0,\ A \vee 1 \rightarrow 1,\ A \vee 2 \rightarrow 2$$

$A \vee 0 \rightarrow 0$ holds as 0 is nothing . $A \vee 1 \rightarrow 1$ is dependant on 1 and therefore does not matter what A is.

3.7 Conjunctive Normal Form

Conjunctive normal form (CNF) is a formula that uses Boolean logic, which is a conjunction of disjunctive literals. In other words, CNF uses the functors $\wedge$ to connect clauses and $\vee$ to connect literals. To create a CNF we must start with the building blocks which are made up of variables. The set of Var which we will call a is a set of variables where $\mathbb{N} \subseteq Var$. As a result this will mean that we cannot get the number 0 in our variables. An example of a CNF would be:

$$(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

The CNF formula is satisfiable if we choose $\neg x_1 = TRUE, x_2 = FALSE, \neg x_3 = FALSE$ as our assignments. This as a result would produce a CNF that is:

$$(TRUE \vee FALSE) \wedge (TRUE \vee FALSE \vee FALSE)$$

The first clause will be true and the second clause will be true and the conjunction of two TRUE's will be true, therefore:

$$(TRUE) \wedge (TRUE) = TRUE$$

Within the CNF formulas, clauses will have to have one literal that holds true using the logical functors in order for the clause to be true. The total formula will therefore be true if all clauses are true. All propositional formulas can be transformed into an equivalent formula that is in a CNF format. The conversion takes place by using rules based on logical equivalences which are:

- Double negative elimination law
- De Morgan's laws
- Distributive law

The *Double negative elimination law* shows that the logical equivalence of an object is the double negation of itself, as

$$P \Leftrightarrow \neg\neg P$$

The *De Morgan's laws* shows the equivalence of a negated set that is expanded into its own negated element, shown as

$$(\neg P \wedge Q) \Leftrightarrow (\neg P \vee (\neg Q)$$

$$\neg(P \vee Q) \Leftrightarrow (\neg P) \wedge (\neg Q)$$

The *Distributive law* shows a valid replacement of variables, as

$$(P \wedge (Q \wedge R)) \Leftrightarrow ((P \wedge Q) \vee (P \vee R)$$

3.8 Converting CNF to clause set logic

In order to transform a CNF $\rightarrow$ CLS we must use three rules to make sure we do not lose anything important. These laws will state that when we acquire the transformed CNF, that it will be equivalent to the CNF at the start. The *Commutative law* dictates that when you swap the positions of the variables they are still the same. For example:

$$A \vee B \Leftrightarrow B \vee A$$

The *Associative law* associative law declares that it does not matter how many variables are grouped. For example:

$$A \vee (B \vee C) \Leftrightarrow (B \vee A) \vee C$$

Idempotents law states that a clause with the same variables is equivalent to the variable. For example:

$$(A \vee A) \Leftrightarrow A$$

To represent a CNF in a set clause, each clause from the CNF is separated and we look at the set by itself out of context of the other clauses so that, $\{\boldsymbol{CL} \in \boldsymbol{CNF}\}$. Each set will remove all duplications of the literal due to the rules of simplification. An example of this would be if we take a given CNF and transform it into a $\boldsymbol{CLS}$:

$$G := (A \vee B \vee B) \wedge (C \vee D \vee \neg D)$$

The propositional formula G will be converted to a set by using the rules of simplification:

$$G := \{\{A, B\} \wedge \{C, D, \neg D\}\}$$

We can now take the set of literals and transform it back into a CNF formula following the Commutative, Associative and Idempotents laws to get a new CNF:

$$G` := (A \vee B) \wedge (C \vee D \vee \neg D)$$

As a result we have produced a new CNF. By using this method we prove a formulas satisfiability which is proven by partial assignment.

3.9 Example of a formula being satisfiable

1. $f := (a \vee b \vee \neg c)$
2. $\varphi : Var(f) \rightarrow \{0, 1\}$
3. $Var(f) = \{a, b, c\}$
4. $\varphi(a) := 1$
5. $\varphi(b) := 1$
6. $\varphi(c) := 0$

The first line notes what the formula function f is. The second line shows that φ maps a variable through total assignment. $Var(f)$ is the set of objects to be assigned a value. Lines 4-6 show the φ mapping of the variables $a \rightarrow 1, b \rightarrow 1, c \rightarrow 0$. These values are arbitrary and are chosen at random. The *TASS* of f is therefore:

$$\text{eval}(\varphi, f) \in \{0, 1\}$$

If the eval is true then it is satisfiable. However if the eval is false then it is unclassifiable. If we compute the formula f through the newly assigned variables, the formula will be

$$f(\varphi) = (1 \vee 1 \vee 1)$$

Using the basic binary functor *disjunction* (see Figure 2) we can now see that the formula will be satisfiable as the whole formula will evaluate to 1.

4 Software Applications

Github is a file sharing hosting service which allows users to interact with open source projects. Github allows a user to make a repository which will store documents. Users can then see each others repository to view their work. The main feature of Github is the "push", "pull" and "fork" operations from a repository. Forking a repository will create a identical version of that users repository on your Github account.The person that is collaborating on the project can make changes to their copy and then push it back to the main repository. In doing so the original user can then see the changes that are made and then sync the documents to a new one. Github allows the user to see what has been deleted and what has been added. For the person collaborating on the project, they can now make pull requests to take a copy of the documents to change them. Git is a program that allows a user to push pull and fork repository's and mainly keep track of the files located on a users system. Git uses the command line bash to execute commands in relation to the git files.

5 Planning

The planning stage of the project is an outline of predicted progress and the problems that can arise that would affect the projects schedule. The key components to look at would be a time-line for when an idea of the progress. This allows one to set enough time to get tasks complete within the projects time-frame, as poor management of time could result in a rush of work near deadlines leading to poor quality of work. The risk Analysis will look at what factors can affect the progress of the project, which can potentially cause the project to fail.

5.1 Time-Line

The main task for the next year is to make a small toy cipher which can be run through a SAT solver using the OKlibrary to attempt to break DES. Once this is complete, the toy cipher can be then modified in its complexity until it is finally a full scale model of a DES.

The planning on the time-line for the initial document was optimistic as with many people when proposing a time line for a project, they try to over achieve, Steady steps must be made in order to understand the project and therefore the time line set is unrealistic. Therefore the time-line has been altered and refined to accommodate this and make more realistic aims that can be achieved with time for reflection.

4th week of first semester	Initial Document
5th week of first semester	Review and reflect the Initial document
6th week of first semester	Build an understanding of the preliminaries that are going to be used
7th week of first semester	Complete a comprehensive understanding of Boolean functions with examples
8th week of first semester	Research Partial assignment and Total assignment
9th week of first semester	Define and explain CNF and conversion of CNF to clause set logic with use of examples
10th week of first semester	Explain with examples how partial assignment and total assignment can be used to satisfy a formula
11th week of first semester	Interim Document
After the January assessments	Presentation at Gregynog
3rd week of second semester	Take feedback from the presentation to improve areas of the project
4th week of second semester	Carry out tests on DES via OKlibrary to check the DES satisfiability
Before Easter interval	Dissertation outline
11th week of second semester	Project Demonstration
11th week of second semester	Complete Dissertation

5.2 Risk Analysis

Risk analysis looks at what things can go wrong internally or externally that would cause the project to either not be complete or cause problems with the time-line thus causing a decrease in the projects quality.

Unrealistic planning
Poor management of time or not keeping up to date with the time-lines deadlines could cause a major delay on the progress on what will be achieved. This would likely cause the project to be rushed and see a decrease in the quality of work produced. The way around this would be to either have a detailed plan with a breakdown of what needs to be done with each attempt to modify the project. This will ensure time management is used as effi-

ciently as possible.

Illness

If I was to come down ill with some sort of illness or injury, this can potentially halt the project, as being too ill to work would be a major issue. Sometimes illness cannot be helped, however steps should be put in place to make sure should this happen, then the project can remain on schedule. This would be down to detailed planning of the project.

Unambiguous Aims

In the initial document, the aims of the project are there for goals to be achieved as part of the success of the project. However some aims may not be achievable due to restricted resources or time. To make sure this does not happen, a review of the resources needed can be made to prevent this from happening, or some possible aims which are similar can be dropped and covered by other aims. More importantly this paper is a theoretical based, and therefore the aims are not permanently set, as a detailed look at a topic can open up opportunity's to study many different areas. If aims were set to be very specific about one topic then it is possible to over complicate the aims. This means that some aims may not be achievable as some theoretical proofs may not be determinable. As a result the aims must be slightly open, to allow the compensation of a new direction to be looked at.

Human error

Human error can occur at any time and can happen to anyone. The results of this can range from small annoyances to major problems. An example of this would be writing down an algorithm wrong and then submitting it into the OKlibrary program, resulting in possible wrong or void results, compromising the integrity of the project. To resolve this, work must be double checked and time must be given to review said work.

Understanding of software

Learning how a program works and how to fully make use of its operations takes time to learn and understand. Even after a long time spent, one may not fully understand it. As a result coming to terms with how to use some of the software outlines in the software application for example the OKlibrary could cause a major set back in both time and opportunity cost.

References

[1] Pascal Junod. Six ways to break des. `http://lasecwww.epfl.ch/memo/memo_des.shtml`. Accsessed: 12/12/2013.

[2] Victor W. Marek. *Introduction to Mathematics of Satisfiability*. Studies in Informatics Series. Chapman & Hall/CRC, 2009.