Will Jones – 635773

Understanding how to attack DES via SAT

2.3 Introduction to Satasfiability

A formula is satisfiable (SAT) if and only if there is a possible interpretation that makes the formula true. A procedure for SAT is correct if every input on which it returns "yes" or "TRUE". This means that the propositional statement is satisfiable and complete if in addition it returns true every satisfiability input [6]. If the no assignment of values exists the formula is FALSE, and so then we say that the formula is unsatisfiable. An example of a satisfiable formula would be "¬A and B" as we can find the assignment of values "A = FALSE" and "B = TRUE" which would make (¬A and B) = TRUE.

A propositional formula is built from using variables(*x*), AND (∧), OR (∨), NOT (¬) and parentheses "()".

A variable is an alphabetic character that represents a number or object which is either subject to change or not specifically known. A literal is a variable that is either positive(*x*) or negative(*x*). The negative literal is the negation of a positive literal noted with a `. A clause is a disjunction of literals (*x* ∨ *x*).

In order to prove satasfiability of a formula, different methods shall be used such as CNF, DNF and set clause logic.

2.3.1 Conjunctive Normal Form

Conjunctive normal form (CNF) is a formula that uses Boolean logic, which is a conjunction of disjunctive of literals. In other words, CNF uses the logical connectives AND to connect causes and OR's to connect literals.

To create a CNF we must start with the building blocks which are made up of variables which we will refer to as Var. The variables are a set of variables where $\mathbb{N} \subset$ Var. This means that we cannot get the number 0 in our variables. We then take a Var complement which is the inverse of a variable and is declared with a dot over the variable, i.e. a`. The complement of A is A`. If A = 1, then A` will = 0, and vice versa. This will now give us an outline of a function Var → literals U literals`.

An example of a CNF would be:

$$(\neg x1 \lor x2) \land (\neg x1 \lor x2 \lor \neg x3)$$

The CNF formula is satisfiable if we choose ¬x1 = TRUE, x2 = FALSE, ¬x3 = FALSE as our variable assignments. This as a result would produce a CNF that is:

(TRUE or FALSE) and (TRUE or FALSE or 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) and (TRUE) = TRUE

Within CNF formulas, clauses will have to have one literal that holds true using the logical connectives in order for the clause to be true. The total formula will therefore be true if all clauses are true.

CNF makes use of special symbols are used which are called logical connectives (∧,∨,¬,⇒,⇔,⊕).

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 equivalence which are:

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

Double negate law

P is equivalent to ¬¬P

De Morgan's laws

¬(P ∧ Q) equivalent to (¬P) ∨ (¬Q)
¬(P ∨ Q) equivalent to (¬P) ∧ (¬Q)

Distributive law

(P ∧ (Q ∨ R)) equivalent to ((P ∧ Q) ∨ (P ∧ R))

2.3.2 Clause logic

A clause is a finite disjunction of literals (AND of OR's of either positive or negative literals). Clauses are written as follows, where the symbols Ci are literals:

C1 ∧ … ∧ Cn

In some cases, clauses are written as sets of literals, so the example above would be written as {l1,…, Cn}. This set is to be interpreted as the disjunction of its elements is implied by the context. It should be noted that the truth evaluation of an empty clause is always false.

In set notation {a, b} {b, a} have no order. When we transform a propositional formula into a clause set we only take one instance of the literal, therefore eliminating duplications. For example if we have the formula (a ∧ ¬ a ∧ a) then we will get the set {a, ¬a}. This is because there are two a variables and so one will be eliminated.

In order to transform a CNF → set clause → CNF we must use 3 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.

Commutative law

When you swap the positions of the variables they are still the same. For example:

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

Associative law

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

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

Idempotence law

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

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

2.3.3 Converting CNF to Cause set logic

To represent a CNF in a set cause, each cause from the CNF is taken and put into a set. Each set will remove all duplications of the literal due to the rules of simplification as stated before. An example of this would be if we take the CNF found below and transform it into a clause set:

$$((A \vee B) \vee C) \wedge (D \vee (\neg D \vee A)$$

The propositional formula will convert to a set as seen below:

$$\{A,B,C,D,\neg D\}$$

We can now take the set of literals and transform it back in a CNF formula following the Commutative, Associative and Idempotence laws to get a new CNF.

$$(A \vee B \vee C \vee D \vee \neg D)$$

As a result we have produced a disjunction of literals as a prepositional formula. By using this method we prove a formulas satasfiability which is proven by partial assignment.

2.3.3.1 Partial assignment and Total assignment

A partial assignment (PA) 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 undefined variables. The task is to define φ x F for partial assignment φ and clause set F.

We therefore want to show the relation of a partial assignment (PASS) and Clause set logic (CSL). This can be denoted as:

$$*: PASS \times CSL \rightarrow CSL$$

This gives us $\top := \emptyset$, so now we can have $\top \in$ CSL. Therefore

$$\varphi * F = \top$$

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

Example of Partial assignment

<> ∈ PASS is an example of the empty partial assignment (as denoted by nothing in the <>, which means empty).

$\langle v \rightarrow \varepsilon \rangle \in$ PASS where $v \in$ var and $\in \{0,1\}$. V is assigned to epsilon which is within the Partial Assignment where v is also found as a member in the Var and is either 0 or 1.

To simplify a case of Partial assignment were some Var are undefined we use the rules of simplification. The rule will assign an output determined on the value as seen in the following examples below:

$$A \vee 0 \rightarrow 0$$

$$A \vee 1 \rightarrow 1$$

$$A \vee 2 \rightarrow 2$$

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

A total assignment (TA) is an assignment to all literals such that $v \in$ TA where $v \in$ var and $\in \{0,1\}$. It should be noted that If a partial assignment has a mapping such that all variables have a assigned value $v \in$ var and $\in \{0,1\}$, then PA will be = TA.

The SAT solver that is going to be used will assign some var but not all.

2.3.4 Satisfying a formula

Formula F is satisfiable if there is an assignment that yields true, given a precondition assignment of truth values to var ($v \in$ var and $\in \{0,1\}$).

We will use var(F) as a set of var. An example of this would be:

$$F := (a \wedge b) \vee \neg c$$

$$\text{Var}(F) = \{a,b,c\}$$

We have stated that var(F) is made up from the set {a,b,c}. We can now apply a map or function so the values are numbers:

$$f : A \rightarrow B$$

The set A is now assigned or transferred into the set B. In other words the set A (domain var(F)) → B(truth values) {0,1}. The set B is a set of truth values of either 0 or 1. To denote this total assignment we use the standard letter φ.

A total assignment for a propositional formula F is a map:

$$\varphi: \text{var}(F) \rightarrow \{0,1\}$$

2.4 Example of a function being satisfiable

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

The first line notes what the function F is. The second line shows that φ maps a variable a value 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 → 1, b → 1, c → 0. These values are arbitrary and are chosen at random.

The total assignment of F is therefore

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

If the eval is true then the it is satisfiable. However if the eval is 0 then it is unsatisfiable.