Understanding how to attack DES via SAT

William Jones - 635773

December 11, 2013

Contents

1 SAT

What is SAT? (brief introduction to the topic(defining definitions and how to build a framework to solve a SAT problem)only to be completed when the main body is finished).

1.1 Sets

A set is a collection of objects. To show that an object x is in the set X we write it is 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 $\mathcal{P}(X)$.

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:

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

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.

Intersection
The Intersection operation is taking two sets and finding the element that is in both set X and Y, $X \cap Y$. Therefore intersection is: $\{x : x \in Y$ and $x \in X\}$.

Union
The Union operation of two sets is finding 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: $\{x : x \in Y$ or $x \in X\}$.

Difference
The difference operation is finding an element that is in set X but not Y, $X \ / \ Y$. Therefore the difference of sets X and Y is: $\{x : x \in Y$ and $x \notin X\}$.
The sets we have been talking so far 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 <> , i.e., <x, y>.
The Cartesian product of set X and Y is $\{<x,y> : x \in X$ and $y \in Y\}$, so that $X \times Y$.
A relation is a subset of the Cartesian product of sets which is written as $R \subseteq X \times X$ [1].

A function relates an input to a output. Functions are denoted by a small "f" followed by its input. A simple example of this would be f(x) = x^2. When x is 4, f(4) = 16.

1.2 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*[1].
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 $l_1 \vee ... \vee l_k$, where each l_j, $1 \leq j \leq k$ is a *literal* [1]. 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 $\boldsymbol{CLS} := \{F \subseteq \boldsymbol{CL}\}$

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

$$Bool = <\{0,1\}, \vee, \wedge, \neg, \Rightarrow, \equiv, 0, 1>$$

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 $\{\varphi \in \boldsymbol{TASS} : \varphi \in \mathrm{Var} \wedge \varphi \in \{0,1\}\}$. 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., $\varphi * = \top$.

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.

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

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

2 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 satasfiable if we choose $\neg x_1$ = TRUE, x_2 = FALSE, $\neg x_3$ = FALSE as our assignments. This as a result would prduce a CNF that is:

$$(\text{TRUE} \vee \text{FALSE}) \wedge (\text{TRUE} \vee \text{FALSE} \vee \text{FALSE})$$

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

$$(\text{TRUE}) \wedge (\text{TRUE}) = \text{TRUE}$$

Within the CND 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

Double negative elimination law
$P \Leftrightarrow \neg\neg P$

De Morgan's laws
$\neg(P \wedge Q) \Leftrightarrow (\neg P) \vee (\neg Q)$
$\neg(P \vee Q) \Leftrightarrow (\neg P) \wedge (\neg Q)$

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

2.1 converting CNF to clause set logic

In order to transform a CNF $\rightarrow$ CLS $\rightarrow$ 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:

$$A \lor B \Leftrightarrow B \lor A$$

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

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

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

$$(A \lor A) \Leftrightarrow A$$

To represent a CNF in a set clause, each clause from the CNF is seperated and we look at the set by itelf 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 $\boldsymbol{CLS}$:

$$G := (A \lor B \lor B) \land (C \lor D \lor \neg D)$$

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

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

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:

$$(A \lor B) \land (C \lor D \lor \neg D)$$

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

2.2 Example of a formula being satisfiable

1. f:= $(a \lor b \lor \neg c)$
2. φ: $Var(f) \to$ {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 objcts to be assigned a value. Lines 4-6 show the φ mapping of the variables $a \to$ 1, $b \to$ 1, $c \to$ 0. These values are arbitary 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 - then it is unclassifiable.

3 computing DES through SAT

4 References