1 Definitions and first concepts.

In this section Boolean formulas will be introduced. We will make a brief introduction to problems with the foundation of math and check ideas of Russell and Whitehead, Hilbert and Bourbaki.

We first start with the basic building blocks, which collectively form what is called the alphabet. Namely,

- Symbols x, y, z for Boolean variables.
- Values 0 and 1, referring to false and true respectively. The set $\{0, 1\}$ will be named as $\mathbb{B}$.
- Boolean Operators:
  - unary: $\neg$
  - binary: $\wedge, \vee, \rightarrow, \oplus, \leftrightarrow$

  We will consider $\wedge$ of greater priority than $\vee$. These operator are defined by theirs truth table:

$\neg$	
0	1
1	0

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

$\rightarrow$	0	1
0	1	1
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

$\leftrightarrow$	0	1
0	1	0
1	0	1

Definition 1.1. A Boolean formula is defined inductively:

- The constants 0 and 1 are constants.
- Every variable is a formula.
- If F is a formula, then $\neg F$ is a formula.
- The concatenation with a symbol of two formulas is a formula too.

Examples of formulas are $x \vee y$ or $x_1 \wedge x_2 \vee (x_4 \vee \neg x_3 \wedge (x_5 \rightarrow x_6) \vee 0)$.

Definition 1.2. Given a set A it has an associated homonyms problem that consists on, given an arbitrary element e check if $e \in A$.

Definition 1.3. An assignment is a function from the set of Boolean formulas to the set of Boolean formulas, on which some variables $\{x_1, ..., x_n\}$ are replaced by predefined constants $\{a_1, ...a_n\}$ respectively. If none of the variables altered by an assignment α are present on the formula F then $\alpha(F) = F$. We denote as $Var(\alpha)$ the set of those variables that receive a value from α.

One can then *apply* an assignment α to a formula F, denoting it by $F\alpha = \alpha(F)$. To describe an assignment we will use a set that pairs each variable to it value, i.e. $\alpha = \{x_1 \rightarrow 1, ..., x_n \rightarrow 0\}$. For example given an assignment $\alpha_0 = \{x_1 \rightarrow 1, x_2 \rightarrow 1, x_3 \rightarrow 0\}$ and $F_0 = x_1 \rightarrow (x_2 \wedge x_4)$ then $F_0\alpha_0 = 1 \rightarrow (1 \wedge x_4) = x_4$.

Definition 1.4. An assignment is said to *satisfies* a formula F if $F\alpha = 1$ and in the case $F\alpha = 0$ it is said to *falsifies* the statement.

Definition 1.5. A formula F is called *satisfiable* if $\exists \alpha : F\alpha = 1$. Otherwise it is called *unsatisfiable*. The set of all satisfiable formulas is denoted as SAT. The problem SAT is the associated problem. An assignment α that satisfies F is called a model an is denoted as $\alpha \models F$.

A formula F such that for every α assignment happens that $F\alpha = 1$ is a tautology. Given two formulas G, F it is said that G follows from F if $G \to F$ is a tautology.

Definition 1.6. A formula F is said to be in conjunctive normal form if is written as:

$$F = C_1 \wedge ... \wedge C_n$$

Where $C_i = (u_{1,i} \vee ... \vee u_{m_i,i})$ and $u_{i,j}$ are literals, that is, variables or negated variables. The set of all formulas in conjunctive normal form is called *CNF*. The set of all formulas in conjunctive normal form such that exists an n all $m_i = n$, is called *N-CNF*. The intersection of these set with the *SAT* set are called *CNF-SAT* y *N-CNF-SAT*.

A formula in *CNF* could be seen as a collection of clauses. The associated problem with *CNF* is straightforward on $O(n)$. The problem that we will investigate is whether a arbitrary formula F have a *sat-equivalent CNF* formula.

We could define an equal relationship on the set of formulas. Let F, G be formulas. Then $F = G$ if it happens that for each α an assignment such that $F\alpha = 1$ then $G\alpha = 1$ and $G\alpha = 1$ then $F\alpha = 1$

Proposition 1.1. The given equal relationship is a equivalence relationship.

Proof. All three properties follows from the equivalent properties on the constants. □

We could define a partial order relation between the formulas. Let F, G be formulas. Then $F \leq G$ if it happens that for each α an assignment such that $F\alpha = 1$ then $G\alpha = 1$.

Proposition 1.2. The given equal relationship is a equivalence relationship.

Proof. As we then could see each class of equivalent as the set of assignment that satisfies all of the clauses, this property arises from the order of inclusion between sets. □

Lemma 1.1. *For every* SAT *formula there is an associated circuit.*

Proof. Every operator can be seen as a gate and every variable as an input. □

Theorem 1.2 (Tseitin)**.** *There is a* 3-CNF *formula on each equivalent class. Moreover, given an element F there is a equivalent formula G in* 3-CNF *which could be done in polynomial time.*

Proof. This result is important because, now we could be able to talk only about *3CNF* formulas. The fact that they are reachable on polynomial time is important because it means it could be done efficiently. Should this be impossible it will not be of much relevance in practice, as we yearn to solve this problem as efficient as possible (in fact, as polynomial as possible).

We will show that for every circuit with n inputs and m binary gates there is a formula in *3-CNF* that could be constructed in polynomial time in n and m. Then, given a formula we will work with it considering it associated circuit.

We will construct the formulae considering variables $x_1, ..., x_n$ that will represents the inputs and $y_1, ..., y_n$ that will represents the output of each gate.

$$G = (y_1) \wedge \bigwedge_{i=1}^{m} (y_i \leftrightarrow f_i(z_{i,1}, z_{i,2}))$$

Where f_i represents the formulae associated to the i-gate, $z_{i,1}, z_{i,2}$ each of the two inputs of the i-gate, whether they be x_- or y_- variables. This formula is not *3-CNF* yet, but for each configuration being f_i a Boolean operator there would be a *3CNF* equivalent.

- $z \leftrightarrow (x \vee y) = \neg(z \vee x \vee y) \vee (z \wedge (x \vee y)) = \neg(z \vee x \vee y) \vee (z \wedge x) \vee (z \wedge y) =$
  $= (\neg z \wedge \neg x \wedge \neg y) \vee (z \wedge x) \vee (z \wedge y) = (\neg z \vee (z \wedge x) \vee (z \wedge y)) \wedge (\neg x \vee (z \wedge x) \vee (z \wedge y)) \wedge$
  $(\neg y \vee (z \wedge x) \vee (z \wedge y)) = (\neg z \vee x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee z)$
- $z \leftrightarrow (x \wedge y) = \neg(z \vee (x \wedge y)) \vee (z \wedge (x \wedge y)) = (z \wedge x \wedge y) \vee (\neg z \wedge \neg x \wedge \neg y) =$
  $((z \vee (\neg z \wedge \neg x \wedge \neg y)) \wedge (x \vee (\neg z \wedge \neg x \wedge \neg y)) \wedge (y \vee (\neg z \wedge \neg x \wedge \neg y))) = (\neg x \vee z) \wedge (\neg y \vee$
  $z) \wedge (\neg z \vee x) \wedge (\neg y \vee x) \wedge (\neg z \vee y) \wedge (\neg x \vee y)$
- $z \leftrightarrow (x \leftrightarrow y) = \neg(z \vee (x \leftrightarrow y)) \vee (z \wedge (x \leftrightarrow y) = \neg(z \vee (\neg x \wedge \neg y) \vee (x \wedge y)) \vee (z \wedge$
  $(\neg x \wedge \neg y) \vee (x \wedge y))) = (\neg z \wedge \neg(\neg x \wedge \neg y) \wedge \neg(x \wedge y)) \vee (z \wedge (\neg x \wedge \neg y) \vee (x \wedge y))) =$
  $(\neg z \wedge (x \vee y) \wedge (\neg x \vee \neg y)) \vee (z \wedge (\neg x \wedge \neg y) \vee (x \wedge y))) = z \vee (\neg x \wedge \neg y) = (\neg x \vee \neg y \vee z) \wedge$
  $(\neg x \vee \neg z \vee y) \wedge (y \vee z \vee x) \wedge (y \vee \neg y \vee x) \wedge (\neg z \vee z \vee x) \wedge (\neg z \vee \neg y \vee x)$
- $z \leftrightarrow (x \oplus y) = z \leftrightarrow (\neg x \leftrightarrow y)$

In the last item we use the third one.

□

This results implies that if we know how to solve *3-CNF* then we will be able to solve 'full'SAT problems. Also, do it in a efficient-fashion.

Definition 1.7. An assignment is called autark for a formula $F \in CNF$ if for every clause $C \in F$ it happens that if $Var(C) \cap Var(\alpha) \neq \emptyset$ then $C\alpha = 1$, in other words it satisfies all clauses that it 'touches'.

The use of this definition is self-evident, as it would simplifies the problem of resolving a *CNF* clause. The strategy would be simple as obvious: try to make every clause positive. These assignment will give simplifications of the problem, and enabling a good method for these search will be useful.

Should it happen that we got an algorithm for autarks clauses, and iterating it, we could find a solution of any given formula. Finding a polynomial algorithm that find whether it exists any non-empty autark formula and provide it, we could be able of proving that NP = P, as we could solve SAT applying this algorithm iteratively. Anyway, trying to find simple autark assignment, i.e. assignment with not many variables, is a good praxis.

Proposition 1.3. We could reduce the SAT-CNF problem to the Autark-Finding problem.

Proof. Suppose that an algorithm such that if it exists any autark it return one of them, and end with an error code otherwise is given.

Given a formula F, if there is not an autark then there is no solution for the SAT problem. If it find an Autark-assignment αthen we apply the same algorithm to $\alpha(F)$. Also, as it happens that $|Var(\alpha(F))| < |Var(F)|$ so we would only apply the algorithm finitely many times. Also,

F will be solvable if, and only if, $F\alpha$ is solvable.

Moreover, as checking if an assigment is autark is linear on the number of clauses, then it made the autark-finding problem NP-Complete(NP-C further on). □

Proposition 1.4. Given $F \to G$ a tautology, there exists a formula I such that $Var(I) = Var(F) \cap Var(G)$ and both $F \to I$ and $I \to G$ are tautologies. It is not known an polynomial algorithm to solve this problem.

Proof. Let $\{x_1, ..., x_k\} = Var(F) \cup Var(G)$ then we will make I by defining its truth table the following way: Given an assigment α:

$$I\alpha = \begin{cases} 1 & \text{if } \alpha \text{ could be extended to an assigment that } \textit{satisfies } F, \\ 0 & \text{if } \alpha \text{ could be extended to an assigment that } \textit{nulifies } G, \\ * & \text{otherwise.} \end{cases}$$

Where * mean that it could be either 0 or 1. This is well defined because if for an arbitrary happend that $G\alpha = 0$ then $F\alpha = 0$.
For every β an assigment such that $Var(\beta) = Var(F) \cup Var(G)$ then if $\beta(F) = 1$ then $\beta(I) = 1$ so $F \to I$ is a tautology. Similarly it can not happend that $I\beta = 1$ and $G\beta = 0$, because the second it will implie that $I\beta = 0$.

For the last part we will show that, should it happend that a polynomial algorithm for interpolation will mean that $NP = P$, as it is done in [1].

TODO □

To get an intuition about the way that unsolvable clausulas are, we gonna state some simple result about combinatorics and resolution. This will give the lector an idea of how these formulas should be.

Firstly, it is easy to break a big clausula on some smallers one, adding one another on this fashion: Supose we got two positive integers n, m such that $m < n$ a clause $x_1 \vee x_2 \vee ... \vee x_n$ we could split it into two parts $x_1 \vee x_2 \vee ... \vee x_{m-1} \vee y, \neg y \vee x_m \vee ... \vee x_n$. Also given the same clausula with a given length n we could enlarge it one variable adding $x_1 \vee ... \vee x_n \vee y$ and $x_1 \vee ... \vee x_n \vee \neg y$. Note that to enlarge a clause from a length m to a length $n > m$ we would generate 2^{n-m} clauses.

Proposition 1.5. Let F be a CNF formula which has exactly k literals, if $|F| < 2^k$ then F is satisfiable.

Proof. Let $n = Var(F)$, it happens that $n > k$. For each clause $C \in F$ there are 2^{n-k} assignment that falsify F, so in total there could be strictly less than $2^k \cdot 2^{n-k} = 2^n$. Therefore it exists an assignment that assign all variables and not falsifies the formula F. □

Proposition 1.6. Let $F = \{C_1, ..., C_n\}$ be a CNF formula. If $\sum_{j=1}^{m} 2^{-|C_j|} < 1$, then F is satisfacible.

Proof. Enlarging clausulas the way it is explained to the maximun length k ans applying the previous ressult. □

Theorem 1.3 (Lovasz Local Lemma)**.** *Let F be a CNF formula. Supose all the clauses has exactly size k. If for every clause $C \in F$ it is true that there are fewer than $2^k - 2$ clauses $C' \in F$ such that $Var(C) \cap Var(C') \neq \emptyset$ then F is satisfiabe*

TODO

Definition 1.8. Let F be a CNF formula. It is said to be minimally unsatisfable if:

- F is unsatisfable.
- $F \backslash \{C\}$ is satisfable $\forall C \in F$.

Proposition 1.7. Minimallly unsatisfable, then $|F| > Var(F)$.

Proof. □

2 Cosas que seguir haciendo

- NP-Completennes.
- Hacer programa para autarks y programa para tranformar en 3CNF.
- Acabar con combinations restrictions.

3 Sources

References

[1] Schöning, U., Torán, J. (2013). The satisfiability problem. Berlin: Lehmanns.