UNIVERSIDAD DE GRANADA

UNDERGRADUATE THESIS

Consistency in Propositional Logic

Author:
Pedro Bonilla Nadal

Supervisor:
Dr. Serafín Moral Callejón

A thesis submitted in fulfillment of the requirements
for the degrees of Computer Engineering and Mathematics

in the

Department of Computer Science and Artificial Inteligence

March 1, 2020

Contents

Chapter 1

Theoretical Introduction

1.1 Definitions and first concepts.

In this section Boolean formulas will be introduced. 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 1.1.1. A Boolean formula is defined inductively:

- The constants 0 and 1 are formulas.
- 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 to.

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

Definition 1.1.3. 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. Analogously, $Var(F)$ will denote the variables present on a formula F.

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. $\alpha = \{x_1 \to 1, ..., x_n \to 0\}$. For example given an assignment $\alpha_0 = \{x_1 \to 1, x_2 \to 1, x_3 \to 0\}$ and $F_0 = x_1 \to (x_2 \wedge x_4)$ then $F_0\alpha_0 = 1 \to (1 \wedge x_4) = x_4$.

Definition 1.1.4. 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 1.1.5. 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 1.1.6. 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*.

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. Equivalently a clause could be seen as a set of literals. The set of all formulas in conjunctive normal form where $|C_i| = N\ i \in 1, ..., n$ is called *N-CNF*. The intersection of these set with the *SAT* set are called *CNF-SAT* y *N-CNF-SAT*.

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 1.1.1. 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 for each α an assignment such that $F\alpha = 1$ then $G\alpha = 1$.

Proposition 1.1.2. The given equal relationship is an 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 given by the inclusion on sets.

□

Lemma 1.1.1. *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 1.1.2 (Tseitin [6]). *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. 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 formula 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 formula associated to the i-gate, $z_{i,1}, z_{i,2}$ each of the two inputs of the i-gate, whether they are 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 *3-CNF* 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 result is important because, now we could be able to talk only about *3-CNF* 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). This results implies that if we know how to solve *3-CNF* then we will be able to solve 'full'SAT problems.

Definition 1.1.7. 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,i s a good praxis.

Proposition 1.1.3. 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 assignment is autark is linear on the number of clauses, then it made the autark-finding problem NP-Complete(NP-C further on). □

Proposition 1.1.4. 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 assignment α:

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

Where * mean that it could be either 0 or 1. This is well defined because if for an arbitrary happens that $G\alpha = 0$ then $F\alpha = 0$.

For every β an assignment 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 happens that $I\beta = 1$ and $G\beta = 0$, because the second it will imply that $I\beta = 0$.

For the last part we will show that, should it happens that a polynomial algorithm for interpolation will mean that $NP = P$, as it is done in [4].

TODO □

1.2 Satisfiability by Combinatorics

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

Firstly, it is easy to break a big clause on some smaller ones, adding one another on this fashion: Suppose 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 clause 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 1.2.1. 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 1.2.2. Let $F = \{C_1, ..., C_n\}$ be a CNF formula. If $\sum_{j=1}^{m} 2^{-|C_j|} < 1$, then F is satisfiable.

Proof. Enlarging clauses the way it is explained to the maximum length k ans applying the previous result. □

Following this idea we could define the weight of a clause $C \in F$ as

$$\omega(C) = 2^{-|C|}$$

being this the probability that a uniform-random assignment violates this clause.

Corollary 1.2.0.1. *For a formula in CNF, if the sum of the weights of the clauses is less than one then the formula is satisfiable.*

Proof. For this task, we will give a probabilistic algorithm, only to prove that it will end with a big probability. Probabilistic (and heuristics) approaches to the problem would prove later on to be really useful. Let F be a CNF formula regard as a clause set. □

Definition 1.2.1. Let F be a CNF formula. It is said to be minimally unsatisfiable if:

- F is unsatisfiable.
- $F \backslash \{C\}$ is satisfiable $\forall C \in F$.

Lemma 1.2.1. *Let F be a set of clauses. If for every subset G of F it holds that $|G| \leq |Var(G)|$, then F is satisfiable.*

Proposition 1.2.3. Minimally unsatisfiable, then $|F| > Var(F)$.

Proof. □

1.3 Lovász Local Lemma

We continue to prove an interesting lemma on the theoretical analysis of satisfiability problem: the Lovász Local Lemma (LLL). This lemma was first proven on 1972 by Erdös and Lovász while they were studying 3-coloration of hypergraphs. Then it was Moser which understood the relationship between this result an constraint satisfaction problem. The SAT could be regard as the simplest of these problems.

This section is going to be based on the works of Moser, Tardos, Lovász and Erdös as a result. As it will be shown LLL is applicable to set sufficient condition for

satisfiability. We will explain the lemma for theoretical purposes and prove the most general version, and give a constructive algorithm to solve a less general statement of the problem. The principal source of bibliography for the whole section would be Moser PhD. Thesis.

The main contribution of Moser's works to this problem is finding an efficient algorithm to find what assignment satisfies the formula, should happen that F is proved satisfiable by the previous theorem. Previously only probabilistic approaches had been successful.

The probabilistic method is a useful method to prove the existence of objects with an specific property. The philosophy beneath this type of demonstration is the following: in order to prove the existence of an object we do not need to give the said object, instead, we could just consider a random object in the space that we consider an prove that the probability is strictly positive. Then we can deduce that an object with that property exists (if it did not probability would be 0). It is not necessary to provide the exact value, bounding it by a constant greater that 0 would be enough.

This technique was pioneered by Paul Erdös. The LLL takes part because is an useful tool to prove lower bounds for probabilities, allowing us to provide the result.

This section will follow this order:

- Present the notation and general expression for the LLL.
- Use the result to prove an interesting property on satisfiability on CNF.
- Prove the general result with the probabilistic result.
- Provide the more concise CNF-result with a constructive algorithm.

1.3.1 First definitions

We will work here with a very specific type of formulas. Let us call a formula F is in k-CNF if it is in CNF and $\forall C \in F, |C| = k$.

Definition 1.3.1. Let C be a clause in F, the neighborhood of C, denoted as $\Gamma_F(C)$ as

$$\Gamma_F(C) = \{D \in F : D \neq C, Var(C) \cap Var(D) \neq \emptyset\}$$

Analogously, the inclusive neighborhood $\Gamma_F^+(C) = \Gamma(C) \cup \{C\}$.

Further on Γ and Γ^+ will respectively denote inclusive or exclusive neighborhood on CNF formulas or graphs

Definition 1.3.2. Two clauses are *conflicting* if there is a variable that is required to be true in one of then and to be false in the other. The graph G_F^* such that there is an edge between C and D iff they *conflict* in some variable.

Definition 1.3.3. Let Ω be a probability space and let $\mathcal{A} = \{A_1, ..., A_m\}$ be arbitrary events in this space. We say that a graph G on the vertex set $\mathcal{A}$ is a *lopsidependency graph* for $\mathcal{A}$ is more likely in the conditional space defined by intersecting the complement of any subset of its non-neighbors. In others words:

$$P\left(A \,\middle|\, \bigcap_{B\in S} \overline{B}\right) \leq P(A) \qquad \forall A \in \mathcal{A},\ \forall S \subset \mathcal{A}\backslash\Gamma_G^+(A)$$

If, instead of requiring the event to be more likely, we require it to be independent (i.e. to be equal in probability) the graph is called *dependency graph*.

1.3.2 Statement of the Lovász Local Lemma

Theorem 1.3.1 (Lovász Local Lema)**.** *Let Ω be a probability space and let $\mathcal{A} = \{A_1, ..., A_m\}$ be arbitrary events in this space. Let G be a lopsidependency graph for $\mathcal{A}$. If there exists a mapping $\mu : \mathcal{A} \to (0, 1)$ such that*

$$\forall A \in \mathcal{A} : P(A) \leq \mu(A) \prod_{B\in\Gamma_G(A)} (1 - \mu(B))$$

then $P\left(\bigcap_{A\in\mathcal{A}} \overline{A}\right) > 0$.

By considering the random experiment of drawing an assignment uniformly, with the event corresponding to violating the different clauses we could reformulate this result. The weight of each clause is the probability of violating each clause. Therefore, we can state a SAT-focused result.

Corollary 1.3.1.1 (Lovász Local Lema for SAT)**.** *Let F be a CNF formula. If there exists a mapping $\mu : F \to (0, 1)$ that associates a number with each clause in the formula such that*

$$\forall A \in \mathcal{A} : \omega(A) \leq \mu(A) \prod_{B\in\Gamma_G^*(A)} (1 - \mu(B))$$

then F is satisfiable.

Proof. To prove the result it would only be necessary to show that Γ^* is the lopsidependency graph for this experiment. Given $C \in F$ and $\mathcal{D} \subset F\backslash\Gamma_{G_F^*}(D)$ (i.e. no $D \in \mathcal{D}$ conflict with C). We want to check the probability of a random assignment falsifying C given that it satisfies all of the clauses in $\mathcal{D}$, and prove that it is at most $2^{-|C|}$.

Let α be an assignment such that it satisfies $\mathcal{D}$ and violates C. We could generate new assignment from α changing any value on $Var(C)$, and they still will satisfy $\mathcal{D}$ (as there are no conflict) so the probability is still at most 2^{-k}.

□

The result that we will prove in a constructive way will be slightly more strict, imposing the condition not only in Γ^* but in Γ^+

Corollary 1.3.1.2 (Constructive Lovász Local Lema for SAT)**.** *Let F be a CNF formula. If there exists a mapping $\mu : F \to (0, 1)$ that associates a number with each clause in the formula such that*

$$\forall A \in \mathcal{A} : \omega(A) \leq \mu(A) \prod_{B\in\Gamma_G(A)} (1 - \mu(B))$$

then F is satisfiable.

In order to get a result easier to check. If $k \leq 2$ the k-SAT problem is polynomial solvable so we will not be interested on such formulas.

Corollary 1.3.1.3. *Let F be a k-CNF with $k > 2$ formula such that $\forall C \in F$ and $|\Gamma_F(C)| \leq 2^k/e - 1$ then F is satisfiable.*

Proof. We will try to use 1.3.1.2. We will define such $\mu : F \to (0,1)$, $\mu(C) = e \cdot 2^{-k}$. Let $C_0 \in F$ be an arbitrary clause.

$$2^{-k} = \omega(C) \leq \mu(C) \prod_{B \in \Gamma_F(C)} (1 - \mu(B)) = e2^{-k}(1 - e2^{-k})^{|\Gamma_F(C)|}$$

With the hypothesis

$$2^{-k} \leq e2^{-k}(1 - e2^{-k})^{2^k/e-1}$$
$$1 \leq e(1 - e2^{-k})^{2^k/e-1}$$

Being famous that the convergence of the sequence $\{(1 - e2^{-k})^{2^k/e-1}\}_k$ to $1/e$ is monotonically decreasing.

□

1.3.3 Nonconstructive proof of 1.3.1

We explain the way Erdös, Lovász and Spencer originally proved the Lemma. This material is from [1] and [5]. The write-up presented here will resemble the one done by [3].

Thorough the proof we will use repeatedly the definition of conditional probability, i.e. for any events $\{E_i\}_{i \in 1,\dots,r}$,

$$P\left(\bigcap_{i=1}^{r} E_1\right) = \prod_{i=1}^{r} P\left(E_i \middle| \bigcap_{j=1}^{i-1} E_j\right)$$

Further on this subsection we will consider Ω to be a probability space and $\mathcal{A} = \{A_1, \dots, A_m\}$ to be arbitrary events in this space, G to be the lopsidependency graph, and $\mu : \mathcal{A} \to (0,1)$ with such that the conditions of the theorem are satisfied. We first prove an auxiliary lemma.

Lemma 1.3.2. *Let $A_0 \in \mathcal{A}$ and $\mathcal{H} \subset \mathcal{A}$. then*

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) \leq \mu(A)$$

Proof. The proof is by induction on the size of $|\mathcal{H}|$. The case $H = \emptyset$ follows from the hypothesis easily:

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) = P(A) \leq^{1.} \mu(A) \prod_{B \in \Gamma^*_G(A)} (1 - \mu(B)) \leq^{2.} \mu(A)$$

Where 1. uses the hypothesis and 2. uses that $0 < \mu(B) < 1$. Now we suppose that $|\mathcal{H}| = n$ and that the claim is true for all $\mathcal{H}'$ such that $|\mathcal{H}'| < n$. We distinguish two cases. The induction hypothesis will not be necessary for the first of them

- When $\mathcal{H} \cap \Gamma^*_G(A) = \emptyset$ then $P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) = 0 \leq P(A)$ by definition of Γ^*_G and $P(A) \leq \mu(A)$ by definition of μ.

- Otherwise we have $A \notin \mathcal{H}$ and $\mathcal{H} \cap \Gamma^*_G(A) \neq \emptyset$. Then we can define to sets $\mathcal{H}_A = \mathcal{H} \cap \Gamma^*_G(A) = \{H_1, ..., H_k\}$ and $\mathcal{H}_0 = \mathcal{H} \backslash \mathcal{H}_A$.

$$P\left(A \middle| \bigcap_{B\in\mathcal{H}} \overline{B}\right) = \frac{P\left(A \cap \left(\bigcap_{B\in\mathcal{H}_A} \overline{B}\right) \middle| \bigcap_{B\in\mathcal{H}_0} \overline{B}\right)}{P\left(\bigcap_{B\in\mathcal{H}_A} \overline{B} \middle| \bigcap_{B\in\mathcal{H}_0} \overline{B}\right)}$$

We will bound numerator and denominator. For the numerator:

$$P\left(A \cap \left(\bigcap_{B\in\mathcal{H}_A} \overline{B}\right) \middle| \bigcap_{B\in\mathcal{H}_0} \overline{B}\right) \leq P\left(A \middle| \bigcap_{B\in\mathcal{H}_0} \overline{B}\right) \leq P(A)$$

Where the second inequality is given by the definition of lopsidependency graph. On the other hand, for the denominator, we can define $\mathcal{H}_i := \{H_i, ..., H_k\} \cup \mathcal{H}_0$.

$$\begin{aligned} P\left(\bigcap_{B\in\mathcal{H}_A} \overline{B} \middle| \bigcap_{B\in\mathcal{H}_0} \overline{B}\right) &= \prod_{i=1}^{k} P\left(\overline{B_i} \middle| \bigcap_{B\in\mathcal{H}_i} \overline{B}\right) \\ &\geq^{3.} \prod_{i=1}^{k} (1-\mu(H_i)) \geq^{4.} \prod_{B\in\Gamma^*_G(A)} (1-\mu(B)) \end{aligned}$$

Where in 3. the induction hypothesis is used, and in 4. is considering that $H_i \in \Gamma^*_G(A)$ Considering now both parts:

$$P\left(A \middle| \bigcap_{B\in\mathcal{H}} \overline{B}\right) \leq \frac{P(A)}{\prod_{B\in\Gamma^*_G(A)} (1-\mu(B))} \leq \mu(A)$$

Where the last inequality uses the hypothesis on μ.

□

proof of the theorem 1.3.1.

$$P\left(\bigcap_{A\in\mathcal{A}} \overline{A}\right) = \prod_{i=1}^{m} P\left(\overline{A_i} \middle| \bigcap_{j=1}^{i-1} \overline{A_j}\right) \geq^{5.} \prod_{i=1}^{m} (1-\mu(A_i))$$

Where in 5. is used 1.3.2 and since $\mu : \mathcal{A} \to (0,1)$ then $P\left(\bigcap_{A\in\mathcal{A}} \overline{A}\right) > 0$.

□

1.3.4 Constructive proof of 1.3.1.2

Moser[3] proves that it exists an algorithm such that it give an assignment satisfying the SAT formula, should it happen that the formula satisfies 1.3.1.1 conditions. This is no a big deal, as a backtrack would be also capable of providing the solution, given that we know its existence. Not so trivial is that it would run in $O(|F|)$. We will show the version of the algorithm shown in [4].

At first sight it is not clear if it terminates. If F verify 1.3.1.1 it is proved that if would end after running Repair at most $O(\sum_{C\in F} \frac{\mu(C)}{1-\mu(C)})$

Algorithm 1 Moser's Algorithm

```
1: C_1, ..., C_m ← Clauses in F to satisfy, globally accessible
2: α ← assignment on Var(F)
3:
4: procedure REPAIR(α, C)
5:     for v ∈ Var(C) do
6:         α(v) = random ∈ {0, 1}
7:     for j := 1 to m do
8:         if (Var(C_j) ∩ Var(C) ≠ ∅) ∧ (C_j α = 0) then
9:             Repair(C_j)
10:
11: Randomly choose an initial assignment α
12: for j := 1 to m do
13:     if α(C_j) = 0 then
14:         Repair(C_j)
```

Chapter 2

Reductions

In order to demonstrate the utility a series of reductions will be developed. This will imply a formal approach to the resolution of the problems, as well as deploying a little theoretical background to some problems when needed. Unalike other chapters this section is original work, although it generality and not being really complicated made it possible to be found on other works (maybe).

2.1 Proofs

2.1.1 Hamiltonian Cycle

By Cook theorem and the ease of checking whether a cycle is a Hamiltonian cycle, it is known that a reduction from the problem of the Hamiltonian Cycle to SAT exists. This theorem is constructive, so it effectively does give a reduction. Nonetheless, this reduction is unmanageable and in order to use SAT-solvers to improve Hamiltonian cycle resolution it would be necessary to improve it. On this subsection an alternative reduction will be proven.

Definition 2.1.1. A Hamiltonian cycle is a cycle that visit every node in a graph. The associated problem is to check, given a graph, whether whether cycle exists.

We will consider the problem of the Hamiltonian cycle of undetected graphs. Therefore an edge would have two sources instead of a source and a target as it is regarded on directed graphs. Prior to the reduction a little lemma will be proven.

Lemma 2.1.1. *Let $G = (V = \{v_1, ..., v_n\}, E = \{e_1, ..., e_m\})$ be a graph. The set $\{e_{i_1}, ..., e_{i_n}\} \subset E$ is a Hamiltonian cycle if, and only if, each vertex is the source to exactly two edges and the path $\{e_{i_1}, ..., e_{i_n}\} \subset E$ is connected.*

Proof. If each vertex is the source of an edge, then every vertex is accessible by an edge. Also, as every vertex has exactly two edges, each connected component of the graph would be a cycle. As the graph is connected there is only one of such components.

□

In order to make the reduction we will represent with Boolean clauses these two condition:

- We will start defining the variables $e_1, ..., e_n$ that will represent if the edge e_i is choose for the path. Also, if a vertex e_i has as sources v_j, v_k then the variables e_{i,v_j} and e_{i,v_k} will be also defined. The first set of formulas to consider will be:

$$e_i \iff e_{i,v_j} \iff e_{i,v_k} \qquad \forall i \in 1, .., m, \forall j, k \in 1, ..., n$$

Note than if e_j does not have as source v_j then $e_{j,v_j} \iff 0$. To ensure that each vertex is the source of exactly two edges we will define these clauses:

$$\bigwedge_{k=1}^{m} \left(\bigwedge_{i=1}^{m} \bigvee_{\substack{j=1 \\ j \neq i}}^{m} e_{j,v_k} \right)$$

In order two ensure that each vertex is source to at least two edges. Then to ensure that there would not be more than two:

$$\bigwedge_{h=1}^{m} \bigwedge_{\substack{i=1 \\ j=1 \\ k=1}}^{n} \neg e_{i,v_h} \vee \neg e_{j,v_h} \vee \neg e_{k,v_h}$$

- To prove the connectivity we will use the connectivity matrix. Henceforth all matrix will be consider as $n \times n$-sized matrix. Given $A = (a_{i,j})$ such that $a_{i,j} = 1$ if, and only if, there is a edge between v_i and v_j, otherwise $a_{i,j} = 0$. Then consider $A^k = (a^*_{i,j})$, it happens that if $(a^*_{i,j}) = 1$ then there is a path of exactly length k. Then to check the connectivity we will define $A' = \sum_{i=0}^{n} A^i$ and defining the formula:

$$(a'_{1,1} \wedge ... \wedge a'_{1,n}) \tag{2.1}$$

Matrix product could be seen as a Boolean operation (for the purpose that we reach): Given $A = (a_{i,j})$, $B = (b_i, j)$ and $C = A \cdot B$ then

$$c_{i,j} = (a_{i,1} \wedge b_{1,j}) \vee ... \vee (a_{i,n} \wedge b_{n,j})$$

As we do not care about the exact value of the sum in A' but only whether $a'_{i,j}$ is greater than 0 we could consider as sum the *or* operation element-wise. This prove that the expression 2.1 is a formula, a bit laborious to do by hand but quite compatible.

It simple to follow that if we could satisfy all the formulas then there would be a Hamiltonian cycle = $\{e_i \in E : e_i = 1\}$ where the second e_i is the variable and the first one is the edge. If no such cycle exists the formulas will be unsatisfiable. Further work to do would consider the implementation and resolution of the problem, and trying to express every formula in CNF.

We have resolve the problem to graph, although the same resolution is available for multigraphs (graph which could have more than one edge with the same sources), as this difference does not affect the property. The next easy results prove this statement.

Bibliography

[1] Paul Erdős and László Lovász. "Problems and results on 3-chromatic hypergraphs and some related questions". In: *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite And Finite Sets, Keszthely (Hungray)*. Citeseer. 1973.

[2] Robin Moser. "A constructive proof of the Lovász local lemma". In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 343–350.

[3] Robin Moser. *Exact Algorithms for Constraint Satisfaction Problems*. Logos Verlag Berlin GmbH, 2013, pp. 17–45.

[4] Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*. Vol. 3. Lehmanns media, 2013.

[5] Joel Spencer. "Asymptotic lower bounds for Ramsey functions". In: *Discrete Mathematics* 20 (1977), pp. 69–76.

[6] Grigori S Tseitin. "On the complexity of derivation in propositional calculus". In: *Automation of reasoning*. Springer, 1983, pp. 466–483.