

Bachelor's Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Hidden variables in mathematical models of quantum structures

Matěj Petr
Open Informatics

May 2016
Supervisor: Prof. Ing. Mirko Navara, DrSc.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Matěj Petr
Study programme: Open Informatics
Specialisation: Computer and Information Science
Title of Bachelor Project: Hidden Variables in Mathematical Models of Quantum Structures

Guidelines:

1. Make a review of literature on (non-)existence of hidden variables in quantum structures.
2. Write an overview of current specialized programs for computing in quantum structures.
3. Write a computer program which, for a quantum structure (orthomodular poset) given by a hypergraph, finds all hidden variables (=two-valued states).
4. Consider possibilities of extension of the program and previous results. E.g., construct a set representation if the quantum structure has sufficiently many two-valued states and try to optimize this representation by omitting as many states as possible. Consider a possibility of improvement of some results, in particular, restriction to rational elements in $L(\mathbb{R}^3)$.

Bibliography/Sources:

- [1] Kalmbach, G.: Orthomodular Lattices. Academic Press, London, 1983.
- [2] Pták, P., Pulmannová, S.: Orthomodular Structures as Quantum Logics. Kluwer Academic Publ., Dordrecht/Boston/London, 1991.
- [3] Mermin, N.D.: Hidden variables and the two theorems of John Bell. Rev. Mod. Phys. 65 (1993), 803-815.
- [4] Navara, M.: Mathematical questions related to non-existence of hidden variables. In: L. Accardi, G. Adenier, C. Fuchs, G. Jaeger, A. Yu. Khrennikov, J. A. Larsson, S. Stenholm (eds.), Foundations of Probability and Physics 5, American Institute of Physics Conference Proceedings, Vol. 1101, New York, 2009, 119-126.

Bachelor Project Supervisor: prof. Ing. Mirko Navara, DrSc.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 14, 2015

Acknowledgement / Declaration

I wish to express my sincere thanks to my thesis supervisor Prof. Ing. Mirko Navara, DrSc. for all his help, advice, time, and comments.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 27. 5. 2016

.....

Abstrakt / Abstract

Cílem této práce je seznámení s teorií skrytých proměnných a stavem programů zabývajících se výpočty na kvantových strukturách. Dále je popsána implementace algoritmu, který pro danou kvantovou strukturu (ortomodulární poset) zjistí, zda-li dovoluje dostatečnou množinu skrytých proměnných (dvouhodnotových stavů).

Klíčová slova: kvantová struktura, ortomodulární poset, pravděpodobnost, stav, skrytá proměnná.

The aim of the thesis is an introduction to the theory of hidden variables and a contribution to programs dealing with computations on quantum structures. Furthermore the implementation of a program, which for given quantum structure (orthomodular poset) determines if the structure allows sufficient amount of two-valued states, is described.

Keywords: quantum structure, orthomodular poset, probability, state, hidden variable.

Contents /

1 Intro	1	References	30
1.1 Thesis organisation	1	A Contents of CD	31
2 Preliminaries	2		
2.1 Motivating examples	2		
2.1.1 States	4		
2.1.2 Hidden variables con- jecture	5		
2.2 Definitions	6		
3 Important results in question of nonexistence of hidden vari- ables	8		
3.1 EPR Paradox	8		
3.2 Gleason's Theorem	8		
3.3 Bell's theorem	8		
3.3.1 Bell inequalities	9		
3.4 Kochen-Specker Theorem	9		
3.4.1 Geometric proof (colouring vectors in \mathbb{R}^3) ..	9		
3.4.2 Peres' proof for $L(\mathbb{R}^3)$ and $L(\mathbb{R}^4)$	10		
3.4.3 Cabello proof for $L(\mathbb{R}^4)$..	11		
4 Programs	12		
4.1 GENER	12		
4.2 BIPOLAR	12		
4.3 COMPARE	12		
4.4 MINRE	12		
5 Implementation of algorithm	16		
5.1 Representing hypergraph	16		
5.2 Speeding up the algorithm	16		
5.3 Input and output	17		
5.4 Code	17		
5.4.1 Package main	17		
5.4.2 Package helper	18		
5.4.3 Package generator	18		
6 Experiments	19		
6.1 Data	19		
6.2 Example of using the program .	19		
6.3 Results	20		
6.3.1 Correctness of the al- gorithm	20		
6.3.2 Speed of algorithm	21		
6.3.3 Time complexity of the algorithm	22		
7 Extension of program	26		
7.1 Example of use	26		
8 Conclusion	29		

Tables / Figures

3.1. Orthogonal triads	11	2.1. Experiment from Example 2.1. ...	3
3.2. Orthogonal tetrads	11	2.2. Greechie diagrams from Ex-	
6.1. Proof of OMP not being con-		amples 2.1. and 2.2.	5
crete	21	3.1. Kochen-Specker diagram	10
6.2. Speed of algorithm	22	4.1. Hasse and Greechie diagrams	
6.3. Speed of faster algorithm	22	of E_6	14
6.4. Speed of the algorithm for		4.2. All two-valued states for E_6 ...	15
$G_{n,k}$ graphs	22	6.1. Example of OMP that is not	
		concrete	21
		6.2. Two-valued probability mea-	
		sures on the pentagon	23
		6.3. Block F_n	24
		6.4. Blocks $G_{n,k}$	24
		6.5. Hypergraph $G_{n,k}$	25

Chapter 1

Intro

Quantum mechanics, formulated at the beginning of 20th century, is one of the most important branches of physics. Quantum mechanics deals with phenomena of the microworld where laws of classical physics do not apply.

Quantum mechanics describes particles by the wave function, a complex function of spacetime. Though we know particles' wave function we do not know everything about its properties. We can only predict probability of values of certain property, eg. position or momentum. Heisenberg uncertainty principle limits accuracy with which we can know values of certain pairs of properties (eg. position and momentum).

Quantum mechanics claims measured properties of particles do not have definite value until they are measured and thus the observed value is created as a result of measurement. Some renowned physicists, eg. Einstein, were convinced that quantum theory is insufficient and would be replaced by so called Hidden Variables Theory.

Irish physicist J.S. Bell however showed that Hidden Variables Theories are not possible.

Premises of Hidden Variable Theory:[1]

Value definiteness: All observables defined for a QM system have definite values at all times.

Non-contextuality: If a QM system possesses a property (value of an observable), then it does so independently of any measurement context, i.e. independently of how that value is eventually measured.[1]

1.1 Thesis organisation

In the second chapter we will introduce basic definitions.

In the third chapter we will review most important results regarding Hidden Variables Theories.

In the fourth chapter we will review programs for investigation of concrete logics.

In the fifth chapter we will discuss implementation of the algorithm and in the following chapter we will discuss testing of the algorithm on various hypergraphs.

In the penultimate chapter we will discuss possibility of extending the program.

In the last chapter we will conclude the thesis.

Chapter 2

Preliminaries

The classic probability model was successful in many tasks, mostly as the basis of statistics. However there are at least two reasons for its revision. Some systems violate the assumptions of the classical theory and require a more general probability model. This brings new mathematical problems worth attention.[2]

Quantum mechanics has been the first field which required a revision of the probability theory. Some events cannot be tested simultaneously due to the uncertainty principle. Therefore there is no reason to assign a probability to their conjunction (disjunction, etc.) if such a phenomenon is not observable. This gives us more freedom in the probabilistic description of the system. Without this modification, the theory did not allow to explain phenomena occurring in quantum physics.[2]

In a classical system, the observable events form a Boolean algebra. The states (2.1.1) are described by a mapping which assigns to each event its probability. So the states may be identified with probability measures.[3]

The logic of quantum mechanics is more general – it is non-distributive. For its system of events, several corresponding algebraical structures were suggested, e.g., orthomodular lattices, orthomodular posets, etc.[3]

2.1 Motivating examples

In this section, we present several physical experiments which demonstrate some quantum phenomena and which are described by simple orthomodular structures. We will refer to them for demonstration of different descriptions and features of orthomodular structures.

Example 2.1. [2][3] Let us assume that we observe a firefly in a box arranged as in Figure 2.1. The firefly can move between the quadrants. Assume it is glowing all the time. An observer at point **A** can distinguish whether the firefly is in the left or in the right side of the box. Similarly the observer at point **B** can tell whether the firefly is in the upper or in the bottom part of the box. In the classical case it would be possible to place two observers **A, B** and distinguish four states corresponding to the presence of the atom in particular quadrants.

In quantum systems, however, a simultaneous observation is often impossible. Measurements are destructive (they change the state of the system irreversibly) eg. a single photon can be observed only once. The same situation, characterized by irreversible changes of the states during measurements, is often found in many other fields, such as sociology, psychology, AI etc. In this example, this phenomenon could be recreated by having only one observer in one of the points. We may choose only one of two possible observations, we cannot perform both at the same time. For the observer in place **A** the observable events form a Boolean algebra $A = \{\mathbf{0}, a, a'^A, \mathbf{1}\}$ where a and a'^A represent the event firefly is in the right side and in the left side of the box respectively, and $\mathbf{0}$ and $\mathbf{1}$ denotes the impossible and sure event respectively. Similarly for the observer in the place **B** the observable events form a Boolean algebra $B = \{\mathbf{0}, b, b'^B, \mathbf{1}\}$ where b and

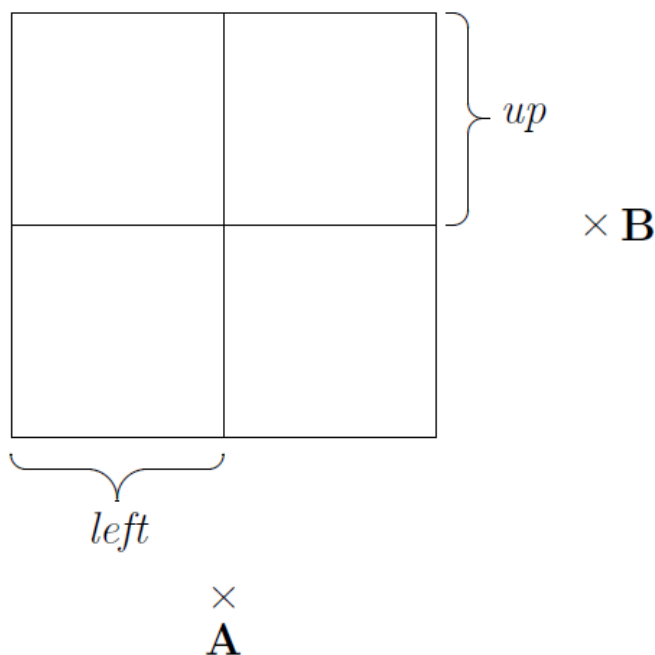


Figure 2.1. Experiment from Example 2.1.[2]

b'^B represent the event firefly is in the upper half and in the bottom side of the box respectively.

It is not possible to observe the conjunction of a and b and other events which are supposed to exist in the classical probability theory. Our system is described by two Boolean algebras, A and B . Their intersection is nonempty, because their bounds (impossible and sure events) are the same: $\mathbf{0}_A = \mathbf{0}_B$, $\mathbf{1}_A = \mathbf{1}_B$. Now we omit the indices.

All observable events form a “logic” $L = \{\mathbf{0}, a, a'^A, b, b'^B, \mathbf{1}\}$ which inherits the ordering and negation of A and B .

Knowing the internal structure, we can consider four internal states of the system. They are described by the results of the observation performed at the states, so we can represent them as mappings from L to the set of truth values, $\{0, 1\}$. Each of these states corresponds to one row in the following table:

s(a)	s(b)
0	0
0	1
1	0
1	1

All the remaining values follow the rules:

(S0) $s(\mathbf{0}) = 0$

(S1) $s(x') = 1 - s(x)$.

All states s on L satisfy (S0) and (S1) and

$s(a) = p, s(b) = q,$

where $p, q \in [0, 1]$ can be chosen arbitrarily.

Example 2.2. [2][3] We take the same system as in Example 2.1. with the only difference the firefly can put out the light. This situation corresponds to a new event, d , with the meaning the firefly is not observed from \mathbf{A} . The events observable from position \mathbf{A} form a Boolean algebra A , isomorphic to 2^3 , having atoms $\mathcal{A}(A) = \{a, d, (a \vee_A d)'^A\}$. Similarly the events observable from \mathbf{B} form a Boolean algebra B with atoms $\mathcal{B}(B) = \{b, d, (b \vee_B d)'^B\}$. All observable events are $L = \{\mathbf{0}, a, b, d, a \vee_A d, b \vee_B d, (a \vee_A d)'^A, (b \vee_B d)'^A, \mathbf{1}\}$ ($d'^A = d'^B$). The *pure states*, states which cannot be expressed as non-trivial convex combinations of different states, are given by the following table:

s(a)	s(b)	s(d)
0	0	0
0	1	0
1	0	0
0	0	1

All states s on L are uniquely determined by the values

$$s(a) = p, s(b) = q, s(d) = r,$$

where $r \in [0, 1]$ is arbitrary and $p, q \in [0, 1 - r]$.

The observable events from Examples 2.1. and 2.2. do not form a Boolean algebra but a Boolean algebras. The basic structure for the description of such systems is an *orthomodular lattice*. It is a bounded lattice L (with bounds $\mathbf{0}, \mathbf{1}$ corresponding to the impossible and the sure event) with a unary operation $' : L \rightarrow L$ (*orthocomplementation*) such that

$$a \leq b \Rightarrow b' \leq a'$$

$$a'' = a$$

$$a \vee a' = \mathbf{1}$$

$$a \vee b = a \vee (a' \wedge (a \vee b))$$

Every orthomodular lattice is a union of Boolean algebras. Elements $a, b \in L$ are *compatible* if they are contained in a Boolean subalgebra of L . Although the lattice operations \wedge, \vee are defined for any couple of elements of an orthomodular lattice, they coincide with the conjunction and the disjunction only for compatible elements. By *quantum structures* we mean not only orthomodular lattices, but also more general structures which are not lattices, orthomodular posets.

Finite (and some infinite) quantum structures admit a representation by hypergraphs called *Greechie diagrams*. Vertices represent *atoms*, i.e., minimal non-zero elements. Edges represent maximal sets of mutually exclusive atoms (which correspond to maximal Boolean subalgebras). The experiments from Examples 2.1. and 2.2. can be described by Greechie diagrams in Figure 2.2.

■ 2.1.1 States

A quantum state of the system can be described by a *probability* measure which is also called a *state* in this context. It is a mapping $s : L \rightarrow [0, 1]$ such that

$$s(\mathbf{1}) = 1,$$

$$s(\bigvee_{i \in \mathbb{N}} a_i) = \sum_{i \in \mathbb{N}} s(a_i) \text{ if } i \neq j.$$

whenever $(a_i)_{i \in \mathbb{N}}$ is a sequence of elements which are mutually *orthogonal*, i.e. $a_i \leq a'_j$.

We demonstrate it on the experiment from Example 2.2. For an observer at \mathbf{A} , the probabilities of elementary events must sum up to one,

$$s(a) + s(a') + s(d) = 1$$

Similarly, for the observer at **B**, we obtain the requirement

$$s(b) + s(b') + s(d) = 1$$

These properties can be easily seen from the Greechie diagrams in Figure 2.2. States on a quantum structure correspond to *states* on their Greechie diagrams, i.e., non-negative evaluations of vertices which sum up to one over each edge. The *state space* (=the set of all states) is closed under convex combinations.[2]

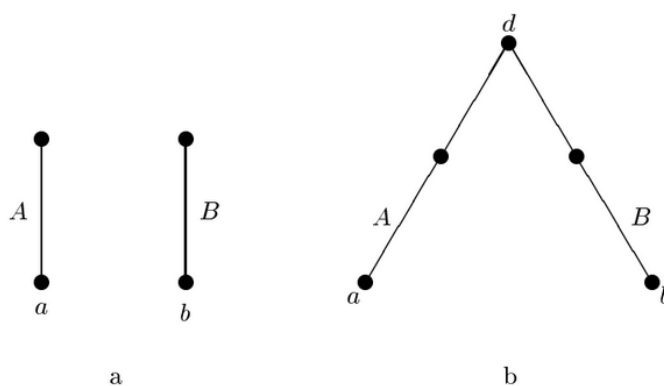


Figure 2.2. Greechie diagrams from Examples 2.1.(a) and 2.2.(b)[3]

2.1.2 Hidden variables conjecture

The quantum theory admits the existence of non-compatible events. As their conjunction cannot be tested, there is no need to assign any probability to it. Nevertheless, there still could be a classical description of a non-classical system, although it would remain unknown. This idea has led to the notion of *hidden variables* which could determine the results of quantum experiments in a classical way. Being not recognizable, they are not in direct contradiction with the limited knowledge in quantum systems.[2]

The idea of hidden variables was strongly defended by Einstein, Podolsky, and Rosen in [4]. This idea was rejected by Heisenberg, von Neumann, and others, but it remained a topic of discussions for several decades. The definite mathematical argument against it was the Gleason's theorem [5] which characterizes probabilities (states) on the lattice of closed subspaces of a Hilbert space. This is the principal example of a quantum structure. Linear subspaces of a Hilbert space H (in case of infinite dimension, only closed subspaces are taken) form an orthomodular lattice, $L(H)$, where

$$\mathbf{0} = 0,$$

$$\mathbf{1} = H,$$

$$A \wedge B = A \cap B,$$

$$A' = \{x \in H \mid \forall y \in A : x \perp y\},$$

$$A \vee B = Lin(A \cup B),$$

where Lin denotes the closed linear hull.

2.2 Definitions

Definition 2.1. An *orthomodular poset (OMP)* is a partially ordered set \mathcal{L} with the largest element 1, the smallest element 0, and unary operation $'$ called *orthocomplementation* on \mathcal{L} satisfying

$$\begin{aligned} x'' &= x \\ x \leq y &\Rightarrow x' \geq y' \\ x \leq y' &\Rightarrow \exists x \vee y \\ x \vee x' &= 1 \\ x \leq y &\Rightarrow \exists z \leq x'(y = x \vee z) \end{aligned}$$

Definition 2.2. Let A be a Boolean algebra. A *state* on A is a mapping $s : A \rightarrow [0, 1]$ such that

$$\begin{aligned} s(1) &= 1, \\ a, b \in A; a \wedge b = 0 &\Rightarrow s(a \vee b) = s(a) + s(b). \end{aligned}$$

The state in the above definition is *finitely additive*.

Definition 2.3. A state is called *two-valued* if it attains only the values 0 and 1.

Definition 2.4. A *hypergraph* is a couple $H = (V, \varepsilon)$ where V is a nonempty set and ε is a covering of V by nonempty subsets of V (i.e. $\bigcup \varepsilon = V$). The elements of V and ε are called vertices and edges respectively.

Definition 2.5. Two hypergraphs $H_1 = (V_1, \varepsilon_1)$ and $H_2 = (V_2, \varepsilon_2)$ are *isomorphic* if there is a one-to-one mapping $i : V_1 \rightarrow V_2$ such that $\varepsilon_2 = \{i(E) : E \in \varepsilon_1\}$.

Definition 2.6. Let Ω be a set. A *concrete logic* on Ω is a collection, \mathcal{E} , of subsets of Ω satisfying

$$\begin{aligned} \Omega &\in \mathcal{E} \\ X \in \mathcal{E} &\Rightarrow \Omega \setminus X \in \mathcal{E} \\ X, Y \in \mathcal{E}, X \cap Y = \emptyset &\Rightarrow X \cup Y \in \mathcal{E} \end{aligned}$$

When we want to refer to the domain, we speak of a concrete logic (Ω, \mathcal{E}) .

If $\{\mathcal{E}_i\}$ is a family of concrete logics on Ω , then $\bigcap \mathcal{E}_i$ is a concrete logic as well. Therefore, for an arbitrary family $\mathcal{G} = \{G_i\}$ of subsets of Ω there exists the least, with respect to inclusion, concrete logic on Ω , $l(\mathcal{G})$, containing all G_i . In this case, G_i are called *generators*, and $l(\mathcal{G})$ is referred to as *generated* by G_i .

Definition 2.7. A set S of states on an OMP \mathcal{L} is called *full (order determining)* if $\forall a, b \in \mathcal{L} : a \not\leq b$ there is a state $s \in S$ such that $s(a) \not\leq s(b)$.

Definition 2.8. The set of states on \mathcal{L} is called $S(\mathcal{L})$.

Definition 2.9. The set of all two-valued states on \mathcal{L} is called $S_2(\mathcal{L})$.

Definition 2.10. An OMP \mathcal{L} is *isomorphic* to a concrete logic iff $S_2(\mathcal{L})$ is order determining.[6]

In this case, we call \mathcal{L} as a *set representable logic*; a *representation* for \mathcal{L} is an arbitrary concrete logic isomorphic to \mathcal{L} as an OMP.

Definition 2.11. Every concrete logic isomorphic to OMP \mathcal{L} is called a *representation* for \mathcal{L} .

Definition 2.12. A representation (Ω, \mathcal{E}) is said to be *minimal* providing Ω is a minimal (under inclusion) full collection of two-valued states.

Definition 2.13. A representation (Ω, \mathcal{E}) of OMP \mathcal{L} is called *total* if $\Omega = S_2(\mathcal{L})$.

Definition 2.14. The minimal nonzero elements of an OMP \mathcal{L} are called *atoms*; we denote by $A(\mathcal{L})$ the set of all atoms in \mathcal{L} . An OMP \mathcal{L} is called *atomistic* provided that $\forall x \in \mathcal{L} (x = \bigvee \{a \in A(\mathcal{L}) \mid a \leq x\})$.

Definition 2.15. Let n be a positive integer, E_n is defined as the OMP whose Greechie diagram is a (proper) n -polygon with three atoms on each side. The automorphism group of E_n is generated by the rotations and symmetries.[7]

Denote by P_0, P_1, \dots, P_{n-1} the vertices of the n -polygon. Obviously a two-valued state on E_n is completely defined by the values on the vertices. So, it is sufficient to indicate the vertices on which a two-valued state equals 1 (in other vertices the state value equals 0 by default).

Definition 2.16. An orthomodular poset \mathcal{L} is *rich* $\iff \forall a, b \in \mathcal{L}, a \not\leq b \exists$ state $s \in S(\mathcal{L}): s(a) = s(b) = 1$.

Definition 2.17. An orthomodular poset \mathcal{L} is *concrete* $\iff \forall a, b \in \mathcal{L}, a \not\leq b \exists$ state $s \in S_2(\mathcal{L}): s(a) = 1, s(b) = 1$. [6]

Chapter 3

Important results in question of nonexistence of hidden variables

3.1 EPR Paradox

EPR paradox is a thought experiment with which its creators, Albert Einstein, Boris Podolsky, and Nathan Rosen, tried to prove that the wave function is not sufficient to the whole description of physical reality. “While we have thus shown that wave function does not provide a complete description of the physical reality, we left open the question of whether or not such a description exists. We believe, however, that such a theory is possible.”[4] Einstein, Podolsky, and Rosen came up with an argument against completeness of quantum mechanics. In other words that there are some concepts of reality which are not described by quantum mechanics. They agreed there must exist deeper layer of reality using some hidden variables that can describe reality in more detailed way than quantum mechanics could. This statement leads to paradoxes. One paradox claims that two particles can interact with each other in a way that would permit to measure both their position and momentum more precisely than what permits Heisenberg Uncertainty Principle under condition measuring one particle immediately influence the other to prevent it. It would mean the particles are exchanging information at a speed faster than the speed of light, that is impossible according to Einstein’s Theory of Relativity.

3.2 Gleason’s Theorem

Definition 3.1. Let $q \in H$, $\|q\| = 1$ on $L(H)$ then the state s_q on $L(H)$, defined by $s_q(Lin(\{y_1, \dots, y_n\})) = \sum_{i=1}^n (q \cdot y_i)^2 = \sum_{i=1}^n \cos^2 \angle(q, y_i)$ for every orthonormal basis (y_1, \dots, y_n) of space H , is called a *vector space*.

Theorem 3.1. Let H be separable Hilbert space of dimension at least three. Then all states on $L(H)$, where $L(H)$ is a lattice of projections on H , are convex combinations of vector states.

A consequence of Gleason’s Theorem is that $L(H)$ allows no two-valued probability measures, thus disproving Hidden Variables Theory.

3.3 Bell’s theorem

John Bell showed that if local hidden variables existed it would be possible to make an experiment with quantum entanglement whose result would satisfy Bell inequalities. If hidden variables do not exist then the Bell inequalities would not be satisfied. It turned out quantum probabilities do not satisfy these inequalities.

3.3.1 Bell inequalities

Let (L, S) be a system where L is an orthomodular σ -lattice and S is a set of states on L . Let $s \in S, a, b, c, d \in L$. [8]

$$\begin{aligned} s(a) + s(b) - s(a \wedge b) &\leq 1 \\ 0 &\geq s(a \wedge b) + s(b \wedge c) + s(c \wedge d) - s(a \wedge d) - s(b) - s(c) \\ s(a) + s(b) + s(c) - s(a \wedge b) - s(a \wedge c) - s(b \wedge c) &\leq 1 \\ s(a \wedge b) + s(b \wedge c) + s(c \wedge d) - s(a \wedge d) - s(b) - s(c) &\geq -1 \end{aligned}$$

3.4 Kochen-Specker Theorem

In a Hilbert space of dimension ≥ 3 there is a set of observables, generalizations of the random variables in quantum structures, for which it is impossible to assign outcomes in a way consistent with quantum mechanics formalism (i.e., in a way that all functional identities satisfied by mutually commuting observables are also satisfied by the values assigned to them in each individual system). [9]

We have spin-1 quantum system that has components in three mutually perpendicular directions S_x, S_y, S_z . We know that the projection of spin-1 system along arbitrary chosen axis can give three results: eigenvalues $-1, 0, 1$. Observables of our interest are squares of S_x, S_y, S_z that can have eigenvalues $0, 1$. Additionally, these squares are commuting and nothing prevents us from measuring them simultaneously. From quantum mechanics we have equation $S_x^2 + S_y^2 + S_z^2 = s(s + 1) = 2$. It follows that two of the values have to be 1 and the third has to be 0. If we could find quantum state in which the result of measuring of any three observables, that are in an orthogonal triad, is not possible to realize with any assignment of 0s and 1s satisfying condition $s(s + 1) = 2$ then we would disprove an existence of Hidden Variable Theory. See the proof below.

3.4.1 Geometric proof (colouring vectors in \mathbb{R}^3)

Find a set of three-dimensional vectors such that it is impossible to colour vectors red(1), blue(0) in such a way that every subset of three mutually perpendicular vectors contains one blue and two red vectors. It can be shown if angle between two vectors of different colour is less than $\tan^{-1} 0.5 \doteq 26.565^\circ$, we can find other vectors that form subset with original two vectors and it is impossible to colour them according to the rules. [10]

We choose unit vector z and mark it blue. We choose vector $a, a = z + \alpha y, 0 < \alpha < 0.5$, laying in plane $y - z$ as the second vector and mark it red.

1. Since vector z is blue, vectors x and y have to be red. Additionally all vectors in plane $x - y$ are red. Arbitrary vector $c = \beta x + y$ has to be red.
2. Similarly all vectors in plane $a - x$ are red. Even vector $d = x/\beta - a/\alpha$ has to be red.
3. Since $a = z + \alpha y, d$ is perpendicular to $c = \beta x + y$. Vectors c and d are red, thus vector $e = c + d$ has to be red.
4. If we express vector e as a sum of explicit forms of vectors c and d , we get $e = (\beta + \beta^{-1})x - z/\alpha$
5. Since $\alpha < 0.5$, then $\alpha^{-1} > 2$. Since $|\beta + \beta^{-1}|$ ranges between 2 and ∞ it is possible to find such β that vector e lays along direction $f = x - z$. Change of sign β will result in the second direction $g = -x - z$.

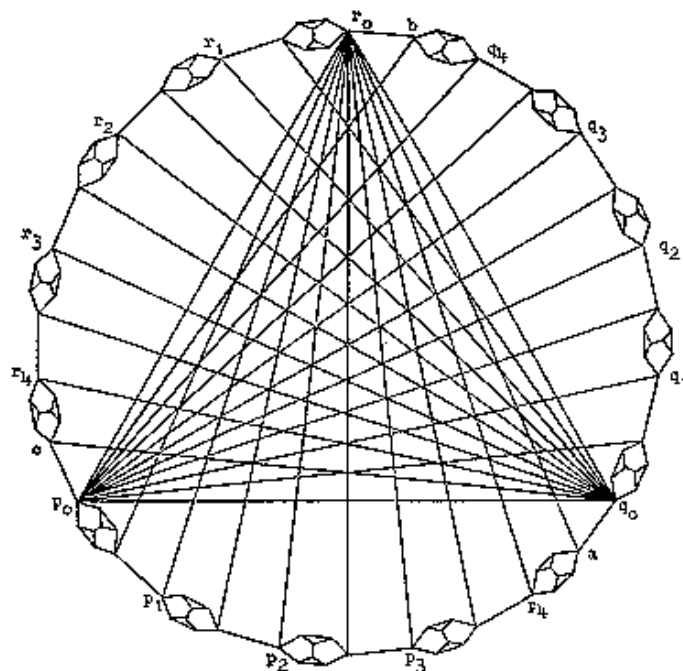


Figure 3.1. Kochen-Specker diagram[11]

6. Vector e is red independently on a choice of parameter β , thus vectors f and g have to be red. Similarly all vectors in plane $f - g$ are red.
7. Vector $z = -0.5f - 0.5g$ lays in plane $f - g$ and thus has to be red, however at the beginning we marked vector z blue. This leads to contradiction.

3.4.2 Peres' proof for $L(\mathfrak{R}^3)$ and $L(\mathfrak{R}^4)$

Let $u_1 \dots u_N$ be a set of vectors forming an orthonormal basis. Let N matrices $P_m = u_m u_m^\dagger$, $m = 1, \dots, N$, be projection operators on vectors u_m . These matrices are commuting and their sum is 1. There exist N different ways of assigning value of 1 to a matrix (i.e. assign 1 to a vector and 0 to $N - 1$ other vectors). We assume several different orthogonal bases which can share some unit vectors. We assume that if a vector is found in more than one basis its value is always the same. This assumption leads to a contradiction as Kochen and Specker proved using 117 vectors.

Peres came up with a set of real three-dimensional rays (vectors) from the center of a cube to its surface in [12]. Vectors end in the center of three sides, six edges, twelve centers of edges, and twelve vertices of the inner cube.

We now assign a value to each ray; 0 (blue) or 1 (red). When we mark one ray blue all perpendicular rays have to be red. We choose a triplet of mutually orthogonal rays (triads) and we mark one ray in each triad blue (in table denoted by bold).

($\bar{1}$ denotes -1, 2 denotes $\sqrt{2}$, $\bar{2}$ denotes $-\sqrt{2}$)

In the Table 3.1. the first, fourth, and the last row contain vectors 100, 021 a $0\bar{1}2$. These rays are red and perpendicular which is contradiction. Proof for four dimensions is analogous. It requires only 24 rays.

In the Table 3.2. the first, third, and fifth row there are vectors 0110, $01\bar{1}0$, $100\bar{1}$, and 1001. These four vectors are red and mutually orthogonal, it again leads to contradiction.

Orthogonal triads	Other perpendicular vectors
001 100 010	110 $\bar{1}\bar{1}0$
101 $\bar{1}01$ 010	
011 $0\bar{1}1$ 100	
1$\bar{1}$2 $\bar{1}12$ 110	$\bar{2}01$ 021
102 $\bar{2}01$ 010	$\bar{2}11$
211 $0\bar{1}1$ $\bar{2}11$	$\bar{1}02$
201 010 $\bar{1}02$	$\bar{1}\bar{1}2$
112 $1\bar{1}0$ $\bar{1}\bar{1}1$	$0\bar{2}1$
012 100 $0\bar{2}1$	$1\bar{2}1$
121 $\bar{1}01$ $1\bar{2}1$	$0\bar{1}2$

Table 3.1. Orthogonal triads

Orthogonal tetrads	Other perpendicular rays
1000 0100 0010 0001	0011 001 $\bar{1}$ 0101 010 $\bar{1}$ 0110 01 $\bar{1}0$
1100 $1\bar{1}00$ 0011 001 $\bar{1}$	$1\bar{1}1\bar{1}$ $1\bar{1}\bar{1}1$ $1\bar{1}11$ $\bar{1}111$
1111 $11\bar{1}\bar{1}$ $1\bar{1}1\bar{1}$ $1\bar{1}\bar{1}1$	$10\bar{1}0$ $100\bar{1}$
1010 $10\bar{1}0$ 0101 010 $\bar{1}$	$11\bar{1}1$
111$\bar{1}$ $11\bar{1}\bar{1}$ $1\bar{1}11$ $\bar{1}111$	1001

Table 3.2. Orthogonal tetrads

■ 3.4.3 Cabello proof for $L(\mathfrak{R}^4)$

Cabello proved non-existence of two-valued state on $L(\mathfrak{R}^4)$ using 18 different vectors in [13]. In every row there is a quadruplet of orthogonal vectors. We assign value of 1 to exactly one vector, the other three will be assigned value of 0. If the vector appears in more than one basis we assume it has constant assignment then it can be shown it is impossible to find two-valued assignment.

$$\begin{aligned}
 v(0, 0, 0, 1) + v(0, 0, 1, 0) + v(1, 1, 0, 0) + v(1, -1, 0, 0) &= 1, (1) \\
 v(0, 0, 0, 1) + v(0, 1, 0, 0) + v(1, 0, 1, 0) + v(1, 0, -1, 0) &= 1, (2) \\
 v(1, -1, 1, -1) + v(1, -1, -1, 1) + v(1, 1, 0, 0) + v(0, 0, 1, 1) &= 1, (3) \\
 v(1, -1, 1, -1) + v(1, 1, 1, 1) + v(1, 0, -1, 0) + v(0, 1, 0, -1) &= 1, (4) \\
 v(0, 0, 1, 0) + v(0, 1, 0, 0) + v(1, 0, 0, 1) + v(1, 0, 0, -1) &= 1, (5) \\
 v(1, -1, -1, 1) + v(1, 1, 1, 1) + v(1, 0, 0, -1) + v(0, 1, -1, 0) &= 1, (6) \\
 v(1, 1, -1, 1) + v(1, 1, 1, -1) + v(1, -1, 0, 0) + v(0, 0, 1, 1) &= 1, (7) \\
 v(1, 1, -1, 1) + v(-1, 1, 1, 1) + v(1, 0, 1, 0) + v(0, 1, 0, -1) &= 1, (8) \\
 v(1, 1, 1, -1) + v(-1, 1, 1, 1) + v(1, 0, 0, 1) + v(0, 1, -1, 0) &= 1. (9)
 \end{aligned}$$

The sum of right sides is 9. The left sides contain each vector twice thus the sum of the left sides is an even number. Therefore this system of equations does not have a solution. The number of vectors with a unit state is odd and even at the same time.

Chapter 4

Programs

I had not found any program that would deal with my problem. However I have discovered programs for investigation of concrete logics made by professor Foat Sultanbekov of Kazan University. All programs had been written in Turbo Pascal. One of the programs, MINRE, could be used as an extension to my program as described in Chapter 7.

4.1 GENER

Let Ω be a finite set and \mathcal{G} a family of subsets of Ω . Program GENER returns the atoms and the blocks of the concrete logic $l(\mathcal{G})$ generated by \mathcal{G} .

4.2 BIPOLAR

This program finds all atoms in a bipolar set of concrete logic.

4.3 COMPARE

This program compares any two collections of sets of nonnegative integers. If these collections are not equal, then their differences will be written in two files.

4.4 MINRE

This program is the most useful regarding the topic of the thesis. It would be possible to use MINRE as an extension of my own program. It finds all minimal and order-determining subsets of Ω . (E is set representable logic and (\mathcal{E}, Ω) is a representation for E .)

The input of the program is file *.dmr. The user should input $card\Omega$ in 3rd line and the number of atoms of \mathcal{E} in 5th line. Atoms of \mathcal{E} should be input in the 9th line.

The code below is an example of the use of the program. The use is illustrated on concrete logic E_6 , an orthomodular poset whose Greechie diagram is a hexagon with three atoms on each side. The two-valued state on E_6 is completely defined by the values on vertices $P_i, i = 0 \dots 5$, of the hexagon. We can describe set $S_2(E_6) = \{a_0, \dots, a_5, b_0, \dots, b_5, c_0, c_1, d_0, d_1, d_2, e\}$ this way:

$$a_k(P_{k+1}) = a_k(P_{k-1}) = 1, (k = 0 \dots 5)$$

$$b_k(P_k) = 1, (k = 0 \dots 5)$$

$$c_k(P_k) = c_k(P_{k+2}) = c_k(P_{k+4}) = 1, (k = 0, 1)$$

$$d_k(P_k) = d_k(P_{k+3}) = 1, (k = 0, 1, 2)$$

and the values on the remaining vertices P_0, \dots, P_5 are zero (in particular, state e vanishes at P_0, \dots, P_5). Indices of vertices are considered modulo 6.

The total representation has 18 elements. We number the states the following way: $b_i := i + 1, (i = 0 \dots 5), e := 7, a_i := 8 + i, (i = 0 \dots 5), c_i := 14 + i, (i = 0, 1), d_i := 16 + i, (i = 0, 1, 2)$. All the two-valued states on E_6 are shown in Figure 4.2. The example file $e_6.dmr$ has the following form.

```

Enter number of all two-valued states on the logic, a natural
number between 2 and 255, on the third line:
18
Enter number of atoms (more than 1):
12
Enter atoms of a logic (elements of atoms are natural numbers
between 1 and 255 at least one interval after each; each atom
should start a line):
1 9 13 14 16
2 8 10 15 17
3 9 11 14 18
4 10 12 15 16
5 11 13 14 17
6 8 12 15 18
3 4 5 6 7 11 12 18
1 4 5 6 7 12 13 16
1 2 5 6 7 8 13 17
1 2 3 6 7 8 9 18
1 2 3 4 7 9 10 16
2 3 4 5 7 10 11 17
Enter logic automorphisms (each permutation on a new line;
with at least one interval after each number):
2 3 4 5 6 1 7 9 10 11 12 13 8 15 14 17 18 16
3 4 5 6 1 2 7 10 11 12 13 8 9 14 15 18 16 17

```

Here each atom represents one vertex of the hexagon. Eg. the first atom tells us that vertex P_i is assigned 1 in following states: b_0, a_1, a_5, c_0 , and d_0 .

MINRE offers also full output which returns all representations including the automorphic representation.

```

Minimal representations of the logic.
Representations of 10 elements:
  C1(2): 1 3 5 8 10 12 14 16 17 18
  C2(1): 7 8 9 10 11 12 13 16 17 18      (2 classes)
Representations of 11 elements:
  C3(1): 1 2 3 4 5 6 14 15 16 17 18      (1 classes)
Representations of 12 elements:
  C4(2): 1 3 5 8 9 10 11 12 13 16 17 18
  C5(6): 1 5 7 8 9 10 11 12 14 16 17 18
  C6(6): 1 3 4 5 6 8 10 14 15 16 17 18    (3 classes)
Representations of 13 elements:
  C7(3): 1 2 4 5 8 9 10 11 12 13 16 17 18
  C8(6): 1 3 4 6 8 9 10 11 13 15 16 17 18
  C9(3): 1 3 4 6 8 10 11 13 14 15 16 17 18  (3 classes)
Representations of 14 elements:
  C10(6): 1 4 5 6 7 8 9 10 11 14 15 16 17 18  (1 classes)
Number of all minimal representations is 36

```

and number of all equivalence classes is 10
 That's all.
 Computing time: 0 hours; 0 minutes; 0,33 seconds.

The program found 10 minimal representations of E_6 .

$$\mathcal{E}_1 = a_{024} b_{024} c_0 \mathbf{d}; \quad \mathcal{E}_2 = \mathbf{a} \mathbf{d} e; \quad \mathcal{E}_3 = \mathbf{b} \mathbf{c} \mathbf{d}; \quad \mathcal{E}_4 = \mathbf{a} b_{024} \mathbf{d};$$

$$\mathcal{E}_5 = a_{01234} b_{04} c_0 e \mathbf{d}; \quad \mathcal{E}_6 = a_{02} b_{02345} \mathbf{c} \mathbf{d}; \quad \mathcal{E}_7 = \mathbf{a} b_{0134} \mathbf{d};$$

$$\mathcal{E}_8 = a_{01235} b_{0235} c_1 \mathbf{d}; \quad \mathcal{E}_9 = a_{0235} b_{0235} \mathbf{c} \mathbf{d}; \quad \mathcal{E}_{10} = a_{0123} b_{0345} \mathbf{c} \mathbf{d} e$$

where a_{12} denotes a_1, a_2 , etc., and \mathbf{a} denotes $a_0 \dots a_5$ and similarly for $\mathbf{b}, \mathbf{c}, \mathbf{d}$. Eg. \mathcal{E}_1 denotes a minimal representation consisting of states $a_0, a_2, a_4, b_0, b_2, b_4, c_0, d_0, d_1, d_2$.

This program could be used as an extension to my program. This is described in Chapter 7.

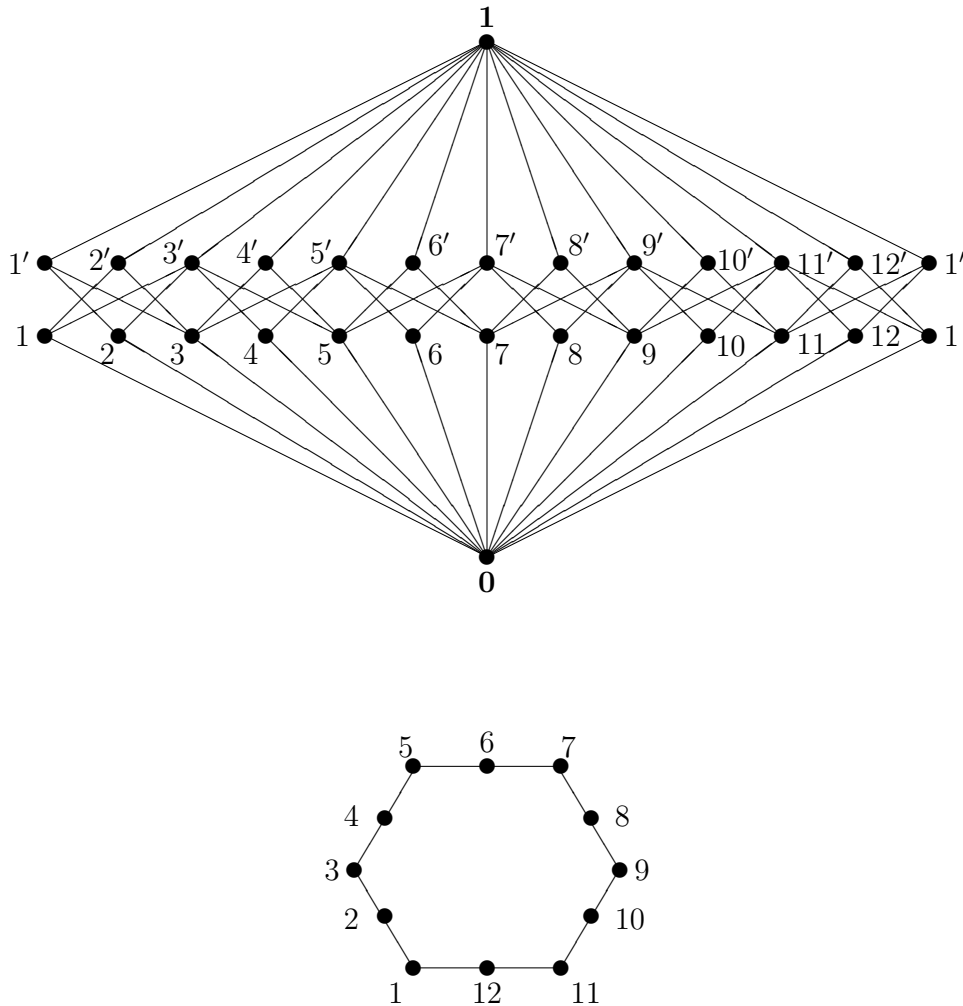


Figure 4.1. Hasse and Greechie diagrams of E_6 . (In the Hasse diagram, elements 1, 1' are marked twice.)

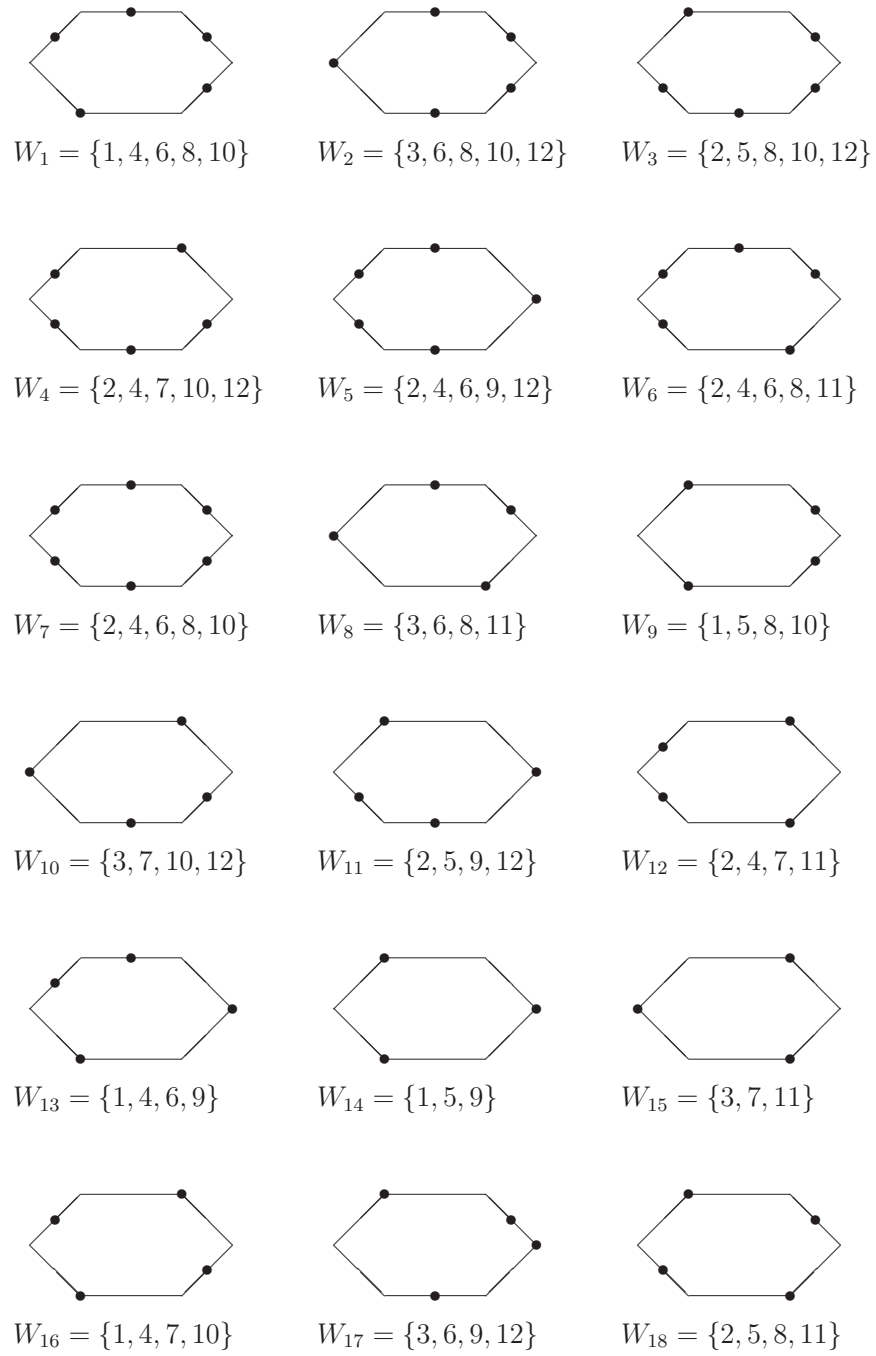


Figure 4.2. All two-valued states for E_6

Chapter 5

Implementation of algorithm

In this section we will focus on implementation of the algorithm.

The algorithm finds two-valued state for each pair of non-orthogonal vertices in OMP represented by hypergraph, ie. it finds if the OMP is concrete. Also the algorithm can find total representation.

5.1 Representing hypergraph

The hypergraph on the input is stored in the textfile. Within the program the hypergraph is respresented by class `Graph` which has attributes `numOfVertices` storing the number of vertices, `numOfEdges` storing the number of edges, `edges` - the list of edges, `automorphism` mandatory attribute storing automorphisms, hashmap `map` where keys are vertices and values are adjacent vertices, and `resultlist` storing two-valued states.

5.2 Speeding up the algorithm

My first version of the algorithm was pretty straightforward. It solved the task for every possible combination of non-orthogonal vertices. This approach was not very fast. For n vertices there are $\frac{n(n+1)}{2}$ combinations of pairs of vertices. This approach produced a lot of duplicate values and since the program uses `java.util.Set`, which does not store duplicates, to store the states there was a lot of useless computation. I improved the algorithm with method `removeDuplicateComb`. This method is called when a two-valued state is found. It removes all combinations of pairs of vertices with value 1 from a list of pairs of non-orthogonal vertices to check. Eg. when the program finds a solution for E_6 (see 4.1) for vertices 1 and 4 - $\{1, 4, 7, 10\}$, then the method removes pairs $\{4, 7\}$, $\{4, 10\}$, and $\{7, 10\}$ from the list since $\{1, 4, 7, 10\}$ is also the solution for the removed pairs above.

Another great improvement was done by rewriting the method `assignZeros` in the class `Solver`. This method is given a number of vertex v with value 1 as one argument and assigns value 0 to all adjacent vertices. Originally this method iterated over all edges in the graph, found every edge with vertex v and assigned 0s to all other vertices in the edge. The speed of the algorithm using this method was tested on several graphs. The results are shown in Table 6.2. and in the sixth column of Table 6.4.

Now upon reading the data the program makes hashmap whose keys are vertices and values are lists of all adjacent vertices, eg. for E_6 (see Figure 4.2.) for key 1 the hashmap stores list $\{2, 3, 11, 12\}$. The hashmap is then used in method `assignZeros`. The method uses vertex v as a key for hashmap and simply assigns value 0 to all vertices adjacent to v . The speed of the algorithm using this improved method was tested on the same graphs as in the previous case. You can see the results in Table 6.3. and in the seventh column of Table 6.4.

5.3 Input and output

The input of the program is the text file determining a hypergraph. The hypergraph should be represented by edges and numbered vertices. Each edge should be on a new line. The user can input automorphisms by writing *automorphism* on the line after last edge and then enter each automorphism on a new line (this can speed up the algorithm for E_n hypergraphs by 40 % as shown in Section 6.2). The output of the program depends on the result. If it is not possible to find the two-valued assignment for some pair of non-orthogonal vertices then the program quits and prints the vertices for which there is not any two-valued assignment. If the orthomodular poset represented by the hypergraph is concrete then the program prints all two-valued states and stores them in a text file.

5.4 Code

The structure of NetBeans project is the following (all files have .java extension):

```
|-- main/
| |-- Main
| |-- Solver
| |-- Edge
| |-- Graph
|-- helper/
| |-- Text
|-- generator/
| |-- Generator
```

5.4.1 Package main

Package `main` contains classes `Main`, `Solver`, `Edge`, and `Graph`. Class `Main` runs the whole program. It is used to receive input and produce output. Class `Solver` finds whether the given orthomodular poset given as a hypergraph is concrete or not. The main method of this class is `solve()` which solves the hypergraph. If we are interested only in two particular non-orthogonal vectors it is possible to call method `initiate(int i, int j)`. This method takes two arguments. Those arguments are integers representing two vertices of the hypergraph. This method finds a two-valued state on the hypergraph subject to $vertice_{i+1} = 1$ and $vertice_{j+2} = 1$. Class `Edge` represents individual edges of the hypergraph. Class `Graph` represents the hypergraph given as input.

Method `solve()` is used for finding all two-valued states on given hypergraph. Firstly this function calls method `getCombinations(ArrayList<Edge> edges, int vertices)` which finds all pairs of vertices and assigns them either 1 for a non-orthogonal pair, or 0 for an orthogonal pair. All pair of vertices are stored in an upper triangular matrix represented by `List<Integer>`. Then the program tries to find a two-valued assignment by calling method `initiate(int i, int j)` for each pair of non-orthogonal vertices. The assignment is stored in `List<Integer> bool`. Each position in the list represents one vertex, it can have values -1 for an unassigned vertex, 0 for a vertex with value 0, or 1 for a vertex with value 1.

The method `initiate` takes two arguments, a pair of non-orthogonal vertices. It assigns them value 1. Each vertex sharing an edge with either of the vertices is assigned value 0. The method then checks if there is an edge with $n - 1$ zeros assigned, where n is the number of vertices in the edge. If so, 1 is assigned to the last unassigned vertex. Then

the recursive function `rec(List<Integer> bool, int start)` is called. This method assigns 1 to the first unassigned vertex, then it assigns 0 to all orthogonal vertices. Then it recursively calls itself. If it fails it returns from recursion and tries to make a different assignment. Thanks to this approach the program will always find some solution if there exists at least one two-valued state.

If the given OMP is not concrete the program will return the first pair of non-orthogonal vertices for which there is no two-valued assignment.

■ 5.4.2 Package helper

Package helper contains class `Text` which handles user interaction.

■ 5.4.3 Package generator

Package generator contains class `Generator`. This class was used for generating hypergraphs for testing purposes. The hypergraphs generated by this class are described in Chapter 6.

Chapter 6

Experiments

6.1 Data

I used two different sets of hypergraphs to test the algorithm. The first set contained hypergraphs with known solutions. This set was used to test correctness of results returned by the program. The second set of graphs was used for testing the speed and limits of the algorithm. For this purpose I used two types of hypergraphs; E_n graphs and graphs from Figure 6.5. generated by class `Generator`. Construction of the graph from Figure 6.5. is shown in Figures 6.3. and 6.4. Firstly block F_n is created. This block has n layers. Secondly block $G_{n,k}$ is created by connecting k blocks F_n . Finally three blocks $G_{n,k}$ are connected together to form the graph from Figure 6.5.

6.2 Example of using the program

We show the use of the program on E_6 . The numbering of vertices is shown in Figure 4.1. Input file `e6.txt` has the following form.

```
1 2 3
3 4 5
5 6 7
7 8 9
9 10 11
11 12 1
```

Each line represents one edge of the hypergraph. The program can be run from NetBeans IDE, or more simply from command line using command `java -jar Program.jar <name_of_file.txt>`. The user can enter the name of the file if it is present in the same folder as `Program.jar` or he can input full path to the file. If the user inputs a correct file the program starts solving the problem. The program can take two optional arguments `-total` and `-set`, `-total` option tells the program to find all two-valued states on given hypergraph, option `-set` tells the program to make set representation of found two-valued states and to generate `*.dmr` file used by program MINRE (for more details and example run of the program with both optional arguments see Chapter 7).

If the solution is not found the program will show for which two non-orthogonal vertices there is no two-valued assignment. If the solution is found the program will output the found two-valued states (only if there are less than 500 two-valued states, otherwise it will print the number of solutions and the time of computation), the number of solutions, and the time of computation in seconds.

```
3, 7, 10, 12
2, 4, 7, 11
3, 6, 8, 11
1, 5, 8, 10
2, 4, 6, 8, 11
```

```

3, 6, 8, 10, 12
2, 4, 7, 10, 12
2, 4, 6, 8, 10, 12
1, 5, 9
1, 4, 6, 8, 10
1, 4, 7, 10
3, 6, 9, 12
2, 5, 8, 10, 12
2, 4, 6, 9, 12
2, 5, 8, 11
1, 4, 6, 9
Solutions found: 16
Computation time: 0.020266802

```

The program found the two-valued assignment for every pair of non-orthogonal vertices. It did not find states W_7 and W_{11} from Figure 4.2. because the two-valued states for all pairs of vertices with value 1 from states W_7 and W_{11} were already found.

6.3 Results

6.3.1 Correctness of the algorithm

The correctness of the algorithm was tested on several graphs with known solutions. Here we list some of them.

Theorem 6.1. Hypergraph E_5 is concrete.

See Figure 6.2. for a proof.

For this hypergraph the program will finish successfully. The two-valued states of E_5 returned by the program are shown below. In each row there is one two-valued assignment. The numbers represents vertices with value 1. We can compare it with the results in Figure 6.2.

```

3, 6, 8, 10
1, 4, 6, 8
1, 5, 8
3, 6, 9
1, 4, 7
3, 7, 10
2, 4, 6, 8, 10
2, 4, 7, 10
2, 5, 9
2, 5, 8, 10
2, 4, 6, 9

```

Theorem 6.2. Orthomodular poset in Figure 6.1. is not concrete.

Proof. Let non-orthogonal vertices 1 and 5 have value 1. All vertices that share an edge with either vertex 1 or 5 must have value 0. Those vertices are $\{2, 3, 4, 6, 7, 11, 12, 14, 20\}$. In the second column of Table 6.1. we can see that the edge $\{6, 12, 17\}$ has two vertices with value 0 and the vertex 17 has no value. Since there has to be exactly one vertex with value 1 in each edge therefore vertex 17 must have value 1. All vertices that share an edge with vertex 17 must have value 0. Those vertices are $\{15, 16, 18, 19\}$. Now there are some edges with two vertices valued 0 and one vertex without value. Those edges are $\{\{13, 14, 15\}, \{3, 24, 18\}, \{8, 2, 16\}, \{10, 4, 15\}\}$. The

vertices $\{8, 10, 13, 24\}$ must have value 1. However this leads to contradiction; in the edge $\{23, 24, 13\}$ there are now two vertices with value 1. QED.

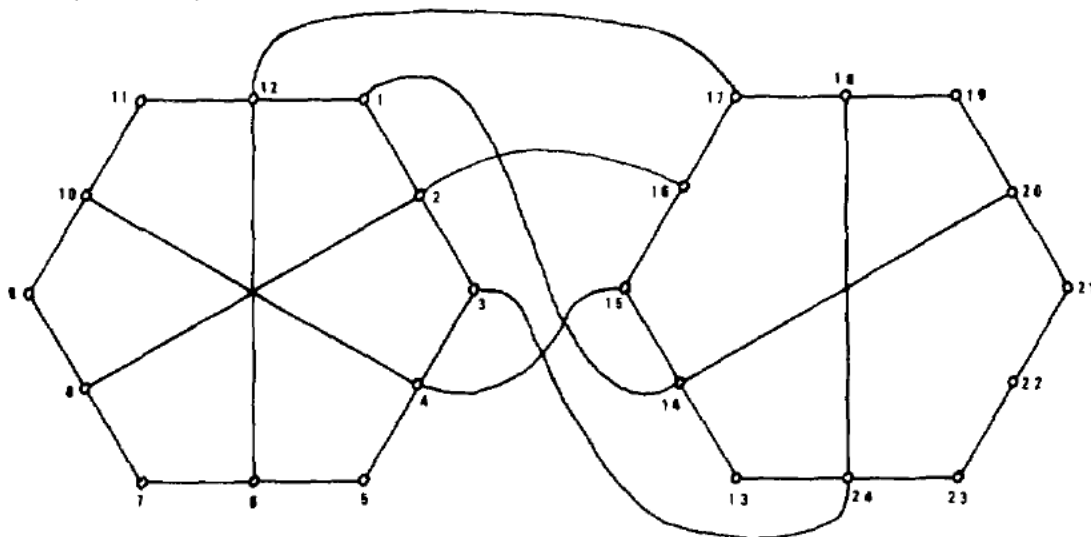


Figure 6.1. Example of OMP that is not concrete. Eg. there is no two-valued assignment for vertices 1 and 5.[14]

Edges	1st step	2nd step	3rd step
1 2 3	100	100	100
3 4 5	00 1	001	001
5 6 7	100	100	100
7 8 9	0??	0??	01?
9 10 11	???	???	?1?
11 12 1	00 1	001	001
13 14 15	?0?	?0 0	100
15 16 17	???	001	001
17 18 19	???	100	100
19 20 21	???	0??	0??
21 22 23	???	???	???
23 24 13	???	???	?11
1 14 20	100	100	100
3 24 18	0??	0? 0	010
6 12 17	00?	00 1	001
8 2 16	?0?	?0 0	100
10 4 15	?0?	?0 0	100

Table 6.1. Proof of OMP from Figure 6.1. not being concrete.

6.3.2 Speed of algorithm

When the number of edges and vertices doubled the speed of the algorithm went down about sixteen times. However the speed of the algorithm does not depend only on the number of edges and vertices but also on the complexity of the hypergraph. Hypergraphs $G_{3,4}$ and E_{100} have similar number of edges and vertices but the graph E_{100} required about six times more time to finish.

When we calculate time per one solution for graphs $G_{3,i}$, where $i = 3, \dots, 7$, we will see that time needed to finish in each instance doubled.

E_n	Time (using automorphism) [s]	Time [s]
E_5	0.0054	0.0036
E_{50}	0.6353	1.118
E_{100}	10.19	18.37
E_{200}	163.5	304.8

Table 6.2. Speed of algorithm

E_n	Time (using automorphism) [s]	Time [s]
E_5	0.0025	0.0058
E_{50}	0.4765	0.8376
E_{100}	7.732	13.20
E_{200}	120.6	215.1

Table 6.3. Speed of faster algorithm

$G_{n,k}$	Vertices	Edges	Pairs of non-orthogonal vertices	Solutions	Time [s]	Time using faster algorithm [s]
$G_{3,3}$	159	96	12561	1191	1.575	0.9169
$G_{3,4}$	204	120	20706	1924	4.411	2.537
$G_{3,5}$	249	144	30876	2990	11.54	6.375
$G_{3,6}$	294	168	43071	4304	31.95	16.49
$G_{3,7}$	339	192	57291	5857	91.34	45.71
$G_{4,3}$	212	122	22366	2507	6.056	2.868
$G_{5,3}$	265	148	34980	4303	15.25	6.769

Table 6.4. Speed of algorithm for $G_{n,k}$ graphs

The use of automorphism improved the speed of the algorithm for E_n graphs greatly. It took only about 60 % of time to solve it. The improved algorithm described in Section 5.2. is about 30 % faster than the previous one.

■ 6.3.3 Time complexity of the algorithm

The algorithm uses backtracking to find the two-valued assignment for a pair of non-orthogonal vertices. Hence the worst case time complexity is $O(n!)$, where n is the number of vertices. This time complexity is reached when the program tries to find the total representation. Therefore it is not recommended to use the program to find the total representation of big hypergraphs. Eg. for E_n the size of total representation can be found by this recursive formulas

$$s_0 = 4,$$

$$s_1 = 7,$$

$$s_i = s_{i-1} + s_{i-2}, i = 2 \dots \infty,$$

where s_0 is a size of total representation of E_3 , s_1 is a size of total representation of E_4 etc. In the best case the function `rec` will need $O(n \cdot k)$ time to find one two-valued assignment, where n is number of vertices and k is number of vertices with value 1. There are $O(n^2)$ pairs of non-orthogonal vertices therefore the resulting time complexity will be $O(n^3)$.

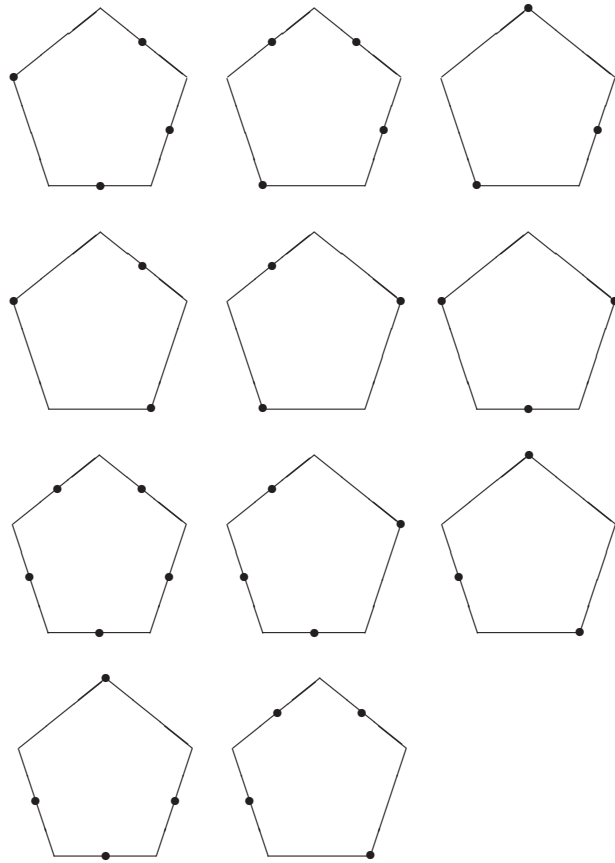


Figure 6.2. Two-valued probability measures on the pentagon.[15] Filled circles indicate probability 1.

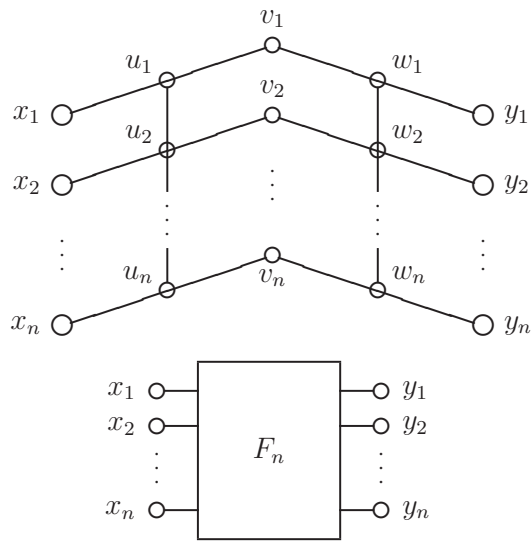


Figure 6.3. Block F_n . The symbol for the block is below.[16]

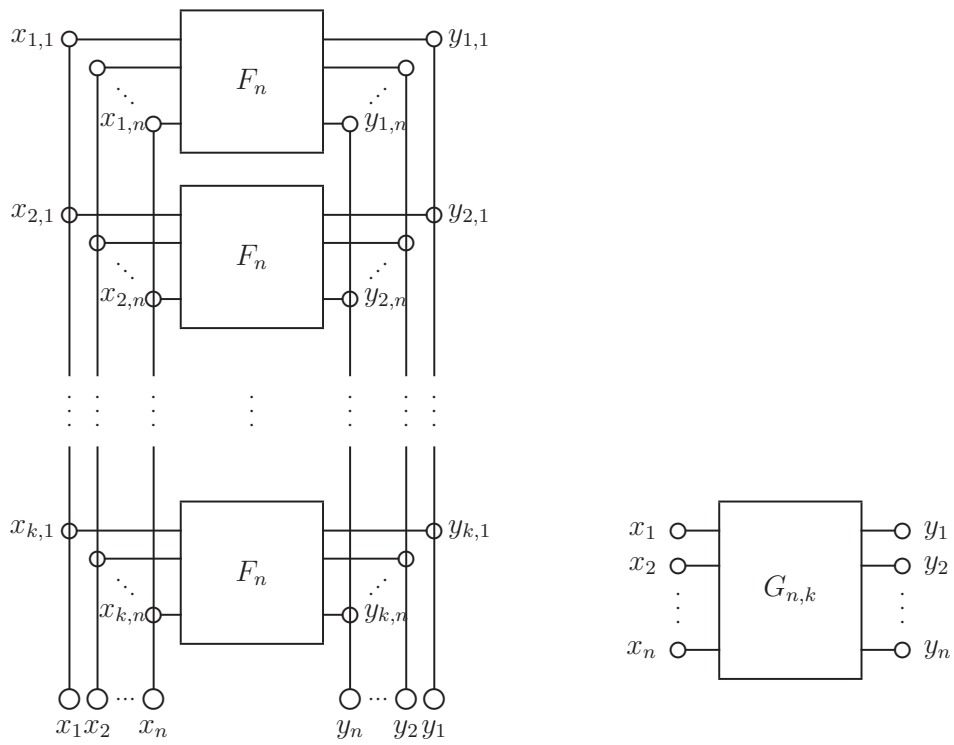


Figure 6.4. Blocks $G_{n,k}$ [16]

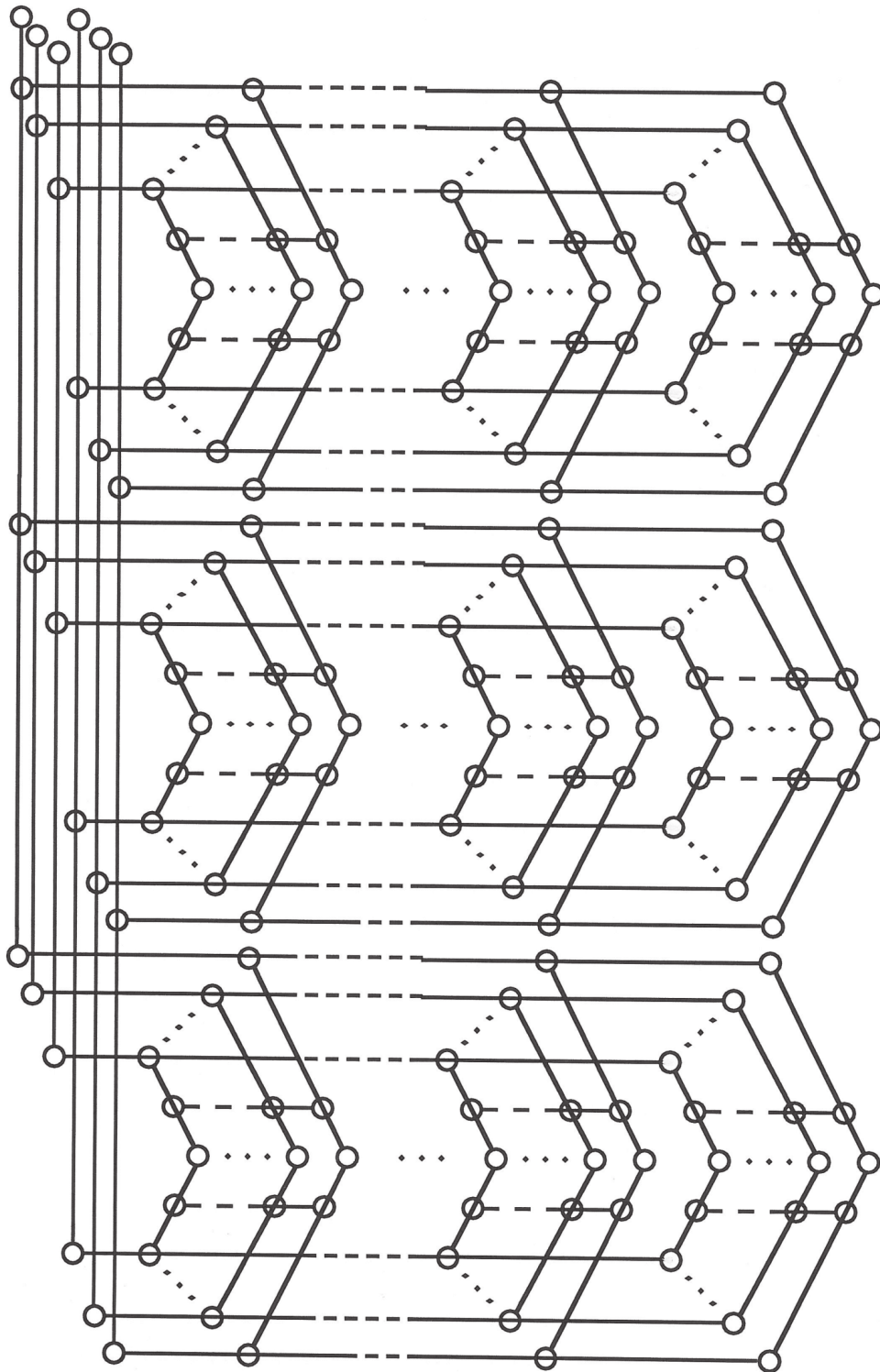


Figure 6.5. Hypergraph $G_{n,k}$ created by connecting hypergraphs from Figure 6.3. and Figure 6.4. used for testing the speed of the algorithm.[16]

Chapter 7

Extension of program

The program can return set representation and generate *.dmr file needed by program MINRE. MINRE can find minimal representation for hypergraphs with at most 255 atoms.

The set representation is generated by method `makeSetRepresentation()` of class `Graph`. This method iterates over each vertex in each two-valued assignment. It stores numbers of assignments in which each vertex has value 1. It is stored in `Map<Integer, List<Integer>>` where the key is one of the vertices of the hypergraph and the value is a list of two-valued assignments in which the vertex has value 1.

7.1 Example of use

For this example we used orthomodular poset E_6 stored in file `e6.txt`. We run the program from command line using command `java -jar Program.jar e6 -total -set`. The program found all two-valued states. The states are listed below.

```
2, 4, 7, 11
3, 7, 10, 12
3, 6, 8, 11
1, 5, 8, 10
2, 4, 6, 8, 11
3, 6, 8, 10, 12
2, 4, 7, 10, 12
2, 4, 6, 8, 10, 12
1, 5, 9
2, 5, 9, 12
1, 4, 6, 8, 10
1, 4, 7, 10
3, 6, 9, 12
2, 5, 8, 10, 12
2, 4, 6, 9, 12
2, 5, 8, 11
1, 4, 6, 9
3, 7, 11
```

On each line there is one state. The numbers listed are vertices with probability 1.

The total representation of E_6 consists of 18 states (see Section 4.4.). See the generated `E_6.dmr` file below.

```
Enter number of all two-valued states on the logic, a natural
number between 2 and 255, on the third line:
18
Enter number of atoms (more than 1):
12
Enter atoms of a logic (elements of atoms are natural numbers
```

between 1 and 255 at least one interval after each; each atom should start a line):

```
4 9 11 12 17
1 5 7 8 10 14 15 16
2 3 6 13 18
1 5 7 8 11 12 15 17
4 9 10 14 16
3 5 6 8 11 13 15 17
1 2 7 12 18
3 4 5 6 8 11 14 16
9 10 13 15 17
2 4 6 7 8 11 12 14
1 3 5 16 18
2 6 7 8 10 13 14 15
Enter logic automorphisms (each permutation on a new line;
with at least one interval after each number):
```

Eg. the first atom 4 9 11 12 17 denotes numbers of assignments in which vertex 1 has value 1.

The program does not enter spatial automorphisms. However if the user wants to use them he can edit the file in Notepad++ or a similar text editor. The output of program MINRE is shown below.

Minimal representations of the logic.

Representation of 10 elements:

```
M1: 1 2 3 4 8 10 12 13 16 17
M2: 4 5 6 7 10 12 13 16 17 18
M3: 1 2 3 9 11 12 13 14 15 16
(3 representations)
```

Representation of 11 elements:

```
M4: 5 6 7 9 11 12 13 14 15 16 18
(1 representations)
```

Representation of 12 elements:

```
M5: 1 2 3 4 8 9 10 11 12 13 15 16
M6: 1 2 3 4 8 9 12 13 14 15 16 17
M7: 4 5 6 7 9 10 11 12 13 15 16 18
M8: 4 5 6 7 9 12 13 14 15 16 17 18
M9: 1 2 3 4 5 6 7 10 12 13 16 17
M10: 1 2 3 4 10 11 12 13 14 15 16 17
M11: 1 2 4 5 6 8 10 12 13 16 17 18
M12: 1 3 4 6 7 8 10 12 13 16 17 18
M13: 2 3 4 5 7 8 10 12 13 16 17 18
M14: 1 2 3 8 9 10 11 12 13 14 16 17
M15: 1 2 5 6 9 11 12 13 14 15 16 18
M16: 1 3 6 7 9 11 12 13 14 15 16 18
M17: 5 6 7 9 10 11 12 13 14 16 17 18
M18: 2 3 5 7 9 11 12 13 14 15 16 18
(14 representations)
```

Representation of 13 elements:

```
M19: 1 2 3 4 6 7 9 10 11 12 13 15 16
M20: 1 3 4 6 7 9 10 11 12 13 15 16 18
M21: 1 2 3 4 5 6 9 12 13 14 15 16 17
M22: 1 2 4 5 6 9 12 13 14 15 16 17 18
M23: 1 2 3 4 6 7 10 11 12 13 15 16 17
M24: 1 2 3 4 5 7 10 11 12 13 14 16 17
M25: 1 2 3 4 5 6 10 12 13 14 15 16 17
M26: 1 2 4 5 6 10 12 13 14 15 16 17 18
M27: 1 3 4 6 7 10 11 12 13 15 16 17 18
M28: 2 3 4 5 7 10 11 12 13 14 16 17 18
M29: 1 2 3 5 7 9 10 11 12 13 14 16 17
M30: 2 3 5 7 9 10 11 12 13 14 16 17 18
(12 representations)
```

Representation of 14 elements:


```
M31: 1 2 4 5 6 8 9 10 11 12 13 15 16 18
M32: 1 3 4 6 7 8 9 12 13 14 15 16 17 18
M33: 2 3 4 5 7 8 9 10 11 12 13 15 16 18
M34: 2 3 4 5 7 8 9 12 13 14 15 16 17 18
M35: 1 2 5 6 8 9 10 11 12 13 14 16 17 18
M36: 1 3 6 7 8 9 10 11 12 13 14 16 17 18
(6 representations)
```

Number of all minimal representations is 36

.

That"s all.

Computing time: 0 hours; 0 minutes; 0,6 seconds.



Chapter 8

Conclusion

Let us summarize the content of the thesis. We have reviewed the most important results in the study of hidden variables. We have reviewed the programs dealing with investigations of concrete logics. Then we introduced the algorithm that will decide whether an orthomodular poset is concrete or not. We have discussed results on various hypergraphs. Finally we have reviewed the possibility of the construction of a set-representation.

We believe that the presented algorithm will be interesting for people involved in the study of quantum structures and will be helpful in practice.



References

- [1] Held, C.: *The Kochen-Specker Theorem*, The Stanford Encyclopedia of Philosophy (Winter 2014 Edition), Edward N. Zalta (ed.), <http://plato.stanford.edu/archives/win2014/entries/kochen-specker/>.
- [2] Navara, M.: *Probability Theory on Quantum and Fuzzy Logics*. Professorial Lectures 6/2005, Czech Technical University, Praha, 2005.
- [3] Navara, M.: *State spaces of orthomodular structures*. Rend. Istit. Mat. Univ. Trieste **31** (2000), Suppl. 1, 143–201.
- [4] Einstein, A., Podolsky, B., & Rosen, N.: *Can quantum-mechanical description of physical reality be considered complete?*. Physical review, **47**(10) (1935), 777.
- [5] Gleason, A. M.: *Measures on the closed subspaces of a Hilbert space*. Journal of Mathematics and Mechanics. **6**(4) (1957), 885–893.
- [6] Gudder, S.P.: *Stochastic Methods in Quantum Mechanics*. North Holland, New York, 1979.
- [7] Sultanbekov, F.: *Set logics and their representations*. International Journal of Theoretical Physics, **32**(11), 2177–2186.
- [8] Pulmannová, S.: *Hidden Variables and Bell Inequalities on Quantum Logics*. Foundations of Physics. **32**(2) (2002), 193–216.
- [9] Macinska, L.: *Kochen-Specker Theorem and Games*. 2007. <http://home.lu.lv/~sd20008/papers/essays/Kochen%20Specker%20%5bpaper%5d.pdf>.
- [10] Mermin, N.D.: *Hidden variables and the two theorems of John Bell*. Rev. Mod. Phys. **65** (1993), 803–815.
- [11] Kochen, S. & Specker, E.P.: *The problem of hidden variables in quantum mechanics*, Journal of Mathematics and Mechanics **17** (1967), 59–87.
- [12] Peres, A.: *Two simple proofs of the Kochen-Specker theorem*. Journal of Physics A: Mathematical and General **24** (1991), 175–178.
- [13] Cabello, A. et al.: *Bell-Kochen-Specker theorem: A proof with 18 vectors*. Phys. Lett. A **212** (1996), 183–187.
- [14] Weber, H.: *There are orthomodular lattices without non-trivial group-valued states: A computer-based construction*. J. Math. Analysis and Appl. **183** (1994), 89–93.
- [15] Svozil, K.: *Quantum Logic*. Springer (1998).
- [16] Mayet, R., Navara, M., Rogalewicz, V.: *Orthomodular lattices with rich state spaces*. Algebra Universalis **43** (2000), 1–30.



Appendix A

Contents of CD

Attached CD contains the following:

- Folder *Program* containing source code of the program
- Folder *Graphs* containing graphs used for testing the program
- Folder *Pdf* containing source files of this pdf
- *Program.jar*
- *petrmat1_bp.pdf*