DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
62

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
62

AHTI PEDER

Superpositional Graphs and Finding the Description of Structure by Counting Method

TARTU UNIVERSITY
PRESS

Institute of Computer Science, Faculty of Computer Science, University of Tartu, Estonia.

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on April 23, 2010 by the Council of the Institute of Computer Science, University of Tartu.

Supervisor:

Prof. Emer. Cand. Phys. Math. Sc. Mati Tombak
University of Tartu
Tartu, Estonia

Opponents:

Prof. Phd. Radomir Stanković
University of Niš
Department of Computer Science
Niš, Serbia

Prof. Emer. Cand. Math. Sc. Leo Võhandu
Tallinn University of Technology
Tallinn, Estonia

The public defense will take place on June 21, 2010 at 13:15 in Liivi 2-403.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 1024–4212
ISBN 978-9949-19-374-5 (trükis)
ISBN 978-9949-19-375-2 (PDF)

Tartu Ülikooli Kirjastus
http://www.tyk.ee/
Tellimus nr. 271

CONTENTS

ACKNOWLEDGEMENTS

This work would not have been possible without the help of many other people. First I would like to express my gratitude to my supervisor Prof. Emer. Mati Tombak and his wife Tiiu for their continuous guidance, interesting suggestions and encouragement during this research.

I express my gratitude to Prof. Raimund Ubar, Jaan Raik and Artur Jutman (their papers were the starting point for this research) and I received valuable suggestions and comments from them.

I also thank to my colleagues (especially Ain Isotamm, Rein Prank and Tiit Roosmaa) in the Institute of Computer Science for their support and help. I wish to thank Prof. Peeter Laud for his valuable advice in improving algorithms 2 and 3, and Reimo Palm for thorougly examining the thesis and giving numerous suggestions for improvements.

Lastly, I thank my family for their patience and support.

LIST OF ORIGINAL PUBLICATIONS

1. Peder, A., Isotamm, A., Tombak, M. A Meta-compiler for propositional formulae. In *Proceedings of the Seventh Symposium on Programming Languages and Software Tools*, Szeged, Hungary (2001), 250–261.

2. Ubar, R., Vassiljeva, T., Raik, J., Jutman, A., Tombak, M., Peder, A. Optimization of structurally synthesized BDDs. In *Proceedings of the Fourth IASTED International Conference on Modelling, Simulation and Optimization*, Kauai, Acta Press (2004), 234–240.

3. Jutman, A., Peder, A., Raik, J., Tombak, M., Ubar, R. Structurally synthesized binary decision diagrams. In *Proceedings of the 6th International Workshop on Boolean Problems*, Freiberg University (2004), 271–278.

4. Peder, A., Tombak, M. Superpositional graphs. *Acta et Commentationes Universitatis Tartuensis de Mathematica*, **13**, (2009), 51–64.

INTRODUCTION

The manipulation of Boolean functions is a fundamental part of computer science, and many problems in the design and testing of digital systems can be expressed as a sequence of operations on Boolean functions. The recent advance in very large-scale integration (VLSI) technology has caused these problems to grow beyond the scope of manual design procedures, and has resulted in wide use of computer-aided design (CAD) systems. The performance of these systems greatly depends on the efficiency of Boolean functions manipulation with, and this is also a very important technique in computer science problems such as artificial intelligence and combinatorics [7].

Binary Decision Diagrams (BDDs) are graph representations of Boolean functions. A BDD is a directed acyclic graph with two terminal nodes, which we call 0-terminal node and 1-terminal node. Each non-terminal node has an index to identify an input variable of Boolean function and has two outgoing edges, called 0-edge and 1-edge [7]. Binary decision diagrams were first introduced by Lee [6] as a data structure for representing Boolean functions. They were further popularized by Akers [1] and Bryant [2]. There are many monographs and textbooks about BDDs, for example [3].

A good data structure is key to efficient Boolean function manipulation. In 1986, Randall Bryant proposed a new data structure called *Reduced Ordered Binary Decision Diagrams* (ROBDDs) in [2]. He showed the simplicity of the manipulations and proved the canonicity of the models, which made it one of the most popular representations of Boolean functions. However, this model is not perfect. Firstly, it suffers from the problem of memory explosion, which limits its usability on large designs. Secondly, it cannot be used as a model for methods that require a certain degree of structural information about the design.

In this thesis we present the advantageous properties of quite an old but not widely known model of *Structurally Synthesized Binary Decision Diagrams* (SSBDDs). They were introduced for the first time in [17] as *Structural Alternative Graphs*. This compact model preserves the structural information about the modeled circuit and utilizes circuit partitioning into a set of subcircuits represented each by its own SSBDD. The algorithms based on SSBDDs are used in the programs of digital diagnostics; SSBDDs present an efficient way of modeling digital systems for simulation purposes [20]. One of the fastest fault simulators in the world is

based on SSBDDs [18, 19].

The most significant difference between the traditional BDDs [2] and SSBDD representations is the method how they are generated. While BDDs are generated by Shannon expansions that extracts the function of the logic circuit, the SSBDD models are generated by a superposition procedure that extracts both: function and data about structural paths of the circuit. Another difference between the classical and the SSBDD approach is that in SSBDDs we represent a digital circuit as a system of BDDs, where for each tree-like subcircuit (fanout-free region) of the circuit a separate SSBDD is generated.

The aim of this thesis is to investigate mathematical properties of SSBDD, to separate the properties of the underlying "skeleton" graphs from the properties of SSBDDs. Necessary skeleton to research SSBDD, *Superpositional Graph* (SPG), was investigated for the first time in [4].

In order to improve the work of diagnostic programs (using on SSBDDs) it is necessary to know the precise properties characterizing the class of SPGs. We need to solve the following problems:

- whether a binary graph G is an SPG;
- whether $G_1, G_2 \in SPG$ are isomorphic;
- whether we can find a sequence of superpositions for $G \in SPG$ that generates it.

The necessary decision problem for decomposing SPG, whether a given binary graph is an SPG, would have been easy to solve if we had known the history of SPG development, i.e., the sequence of superpositions that was used for generating the graph. But we do not have this sequence. Thus, the goal is to develop the necessary and sufficient properties to describe SPGs without using the superposition. Hopefully a better knowledge about SPGs will help us solve the three above-mentioned problems. The development of the methodology presented in this thesis was initiated by the fact that a description of SPGs with *classical properties* could not be found, the set of the found properties turned out to be incomplete.

Further, to find similar structures we tried to count n-node SPGs and found from [10] that the received sequence coincides with the beginning of the sequence of large Schröder numbers. In Section 2.1 we prove that this is not a coincidence: we establish a bijection between well-formed bracketings and superpositional graphs. However, this bijection did not help us very much – we could not translate any of the problems described by large Schröder numbers into the theory of SPGs. We can use only the integer sequence as a starting point of our search for the solution of the problem.

Our next goal is to describe the characteristic set of properties of the class SPG knowing that the set must generate the detected sequence of integers. One possibility for doing it is by means of propositional formulae [12]. We try to find a family

of propositional formulae $\mathcal{F}_n$, which depends on parameter n so that $\#SAT(\mathcal{F}_n)$ is the number of n-node superpositional graphs. Choosing the propositional variables in such way that every assignment corresponds to some binary graph, we can interprete the formulae as the properties of the binary graphs.

We try to describe our structure by the family of propositional formulae and count the number of models for small values of the family parameter. The goal is to refine the logical description until the received integer sequences coincide. After this we "translate" the logical description into a mathematical one and prove it. The main idea of this methodology is to find the models that do not fit with current approximation of the description of the structure and, stepwise, refine the logical description.

A bottleneck of the method could be the fact that we must calculate manually or write a separate program for each of the found approximations, which constructs a *classical* propositional formula for every value of the family parameter. The problem will be solved by the translator for metaformulae [8, 25], which translates the description of the family of propositional formulae to a propositional formula corresponding to the value of the parameter n. By using this translator we will find sets of properties that generate the integer sequences which are gradually closer to the correct integer sequence.

The main results of this thesis are:

- the methodology for finding a description by counting method;
- the grammar necessary for the description of the family of formulae;
- the translator, which, according to the value of the parameter, translates the family of propositional formulae into the traditional propositional formula;
- the description of the class of superpositional graphs.

The thesis is organized as follows. In Chapter 1 we discuss the SSBDD model and give some properties of SPGs.

In Chapter 2 we see that the properties, found in Theorem 1.2, are not sufficient and show how to find the description of structure by counting method. We illustrate expressions of properties by means of propositional formulae with a few examples.

Chapter 3 contains the descriptions of the language and the translator for families of propositional formulae. Subsequently we illustrate how the translator works.

In Chapter 4 we find the first family of formulae, which describes the class SPG. We can see that the found properties are not easily comparable with the properties of *classical* graph theory. On the second try (Chapter 5), we change the domain of the formula that we are looking for and we succeed in finding a "sensible" formula.

In Chapter 6 we prove by traditonal graph-theoretical methods that the lastly found set of properties describes exactly the class SPG.

Finally, in Chapter 7 we give the solutions to the problems raised.

In conclusion we find that the existence of the created translator is essential in the process of describing superpositional graphs. Thus the grammar and the process of finding the family of propositional formulae are thoroughly analyzed in the thesis.

CHAPTER 1

STRUCTURALLY SYNTHESIZED BINARY DECISION DIAGRAMS

In this chapter we describe a special class of BDDs called Structurally Synthesized BDDs (SSBDD). The results described in Section 1.2 are the author's contribution, the remaining part is reproduction from [4] and written by coauthors. Nevertheless, this part is included in the thesis in order to explain the background.

The most significant difference between the traditional BDDs and SSBDDs representations is the method how they are generated. While BDDs are generated by Shannon expansions [3], which extract the function of the logic circuit, the SSBDD models are generated by a superposition procedure that extracts both, function and data on structural path of the circuit. Another difference between the classical and the SSBDD approach is that in SSBDDs we present a digital circuit as a system of BDDs, where a separate SSBDD is generated for each fanout-free region (FFR) of the circuit.

Next we will show that the special properties of SSBDDs are highly useful in fault modeling of digital circuits. Experimental results of fault simulation on ISCAS85 benchmark circuits were carried out [4] where speedup of 2 to 7 times in comparison to traditional gate-level descriptions was achieved.

1.1 Definitions

Let us denote $B = \{0, 1\}$. A *Boolean function* is a mapping $f : B^n \rightarrow B$. Let $F(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ be a Boolean function. We denote

$$F_{x_i} = F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n),$$

$$F_{\bar{x}_i} = F(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n).$$

A *Boolean derivative* is

$$\frac{\partial F}{\partial x_i} = F_{x_i} \oplus F_{\bar{x}_i}.$$

Every satisfying assignment of the Boolean derivative has the following property: changing the value of the variable x_i changes the value of the function.

In this chapter we are using three different formalisms to represent Boolean functions: circuits, propositional formulae and decision diagrams.

Propositional formula with variables $x_1, \ldots, x_n$, constants 0 ($false$) and 1 ($true$) on the basis $\{\vee, \&, \neg\}$ is defined inductively as follows:

1° every variable and constant is a propositional formula;

2° if F is a propositional formula, then $(\neg F)$ is a propositional formula;

3° if P and R are propositional formulae, then $(P \& R)$ and $(P \vee R)$ are propositional formulae.

The priority of Boolean connectives is $\vee, \&, \neg$ (the highest priority). We allow omitting parentheses if there is no confusion in determining the structure of subformulae. We say that x and $\bar{x}$ are *literals* for variable x. A propositional formula F is a disjunctive normal form (DNF), if $F = \bigvee_{i=1}^{p} T_i$, where $T_i = \bigwedge_{j=1}^{k_i} l_j$ are terms and l_j are literals.

A term T *subsumes* a term S if S contains all literals of T. Subsumed terms can be removed from DNF without altering the function it represents. We denote the fact that $F(\alpha_1, \ldots, \alpha_n) = 1$ for $\alpha = (\alpha_1, \ldots, \alpha_n) \in B^n$ by $\alpha \vdash F$.

By $\#SAT(F)$ we denote the number of satisfying assignments of the Boolean function F.

A *binary graph* is an oriented acyclic connected graph with root and two terminals (sinks), 0 and 1. Every internal node v has two sucessors: $high(v)$ and $low(v)$. Therefore, an edge $a \to b$ is 0-*edge* (1-*edge*) if $low(a) = b$ ($high(a) = b$). Binary graphs are skeletons of binary decision diagrams (BDD): a BDD is a binary graph, in which internal nodes are labelled by propositional variables, the terminal 1 is labelled by the truth value 1 and the terminal 0 by the truth value 0. We denote the label of the node v by $label(v)$.

Let D be a binary decision diagram with variables $x_1, \ldots, x_n$. Every vector $\alpha \in B^n$ *activates* a path $p(\alpha) = p_0, \ldots, p_k$ in D from the root to a terminal node: if $\alpha \vdash label(v_i)$, then $v_{i+1} = high(v_i)$ else $v_{i+1} = low(v_i)$. The Boolean function $f_D(x_1, \ldots, x_n)$, represented by D, is defined as follows: $f(\alpha) = 1$ iff the path, activated by α, ends in terminal 1.

A BDD is called *free* if each variable is encountered at most once on each path from the root to the terminal vertex. Free BDDs have useful computing properties: counting the number of true assignments and testing the satisfiability can be done

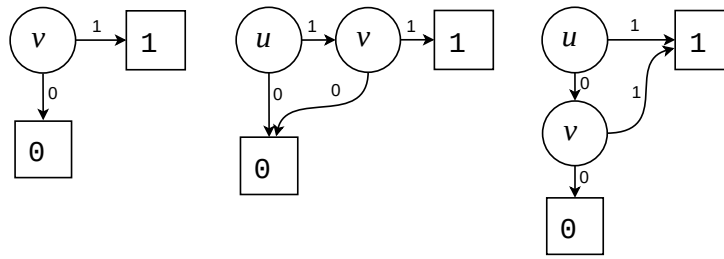


Figure 1.1: Binary graphs A, C, and D.

in time, linear in the number of nodes of the free BDD [2, 3]. We have to pay for this with the size of free BDD – there exist functions for which the size of the minimal free BDD is exponential in the number of variables, in the formula size and in the circuit size.

1.2 Superpositional Graphs: Definitions and Properties

Definition 1.1. A *superposition* of graph E into graph G instead of an internal node v, denoted by $G_{v \leftarrow E}$, is a graph, which we get by deleting v from G and redirecting all edges, pointing to v, to the root of E, all edges of E pointing to terminal 1, to the node $high(v)$ and all edges, pointing to the terminal 0, to the node $low(v)$.

Let A, C, and D be binary graphs, whose descriptions are shown in Figure 1.1.

Definition 1.2. A *class of superpositional graphs* (SPG) is defined inductively as follows:

1° graph $A \in SPG$;

2° if $G \in SPG$ and v is an internal node of G, then $G_{v \leftarrow C} \in SPG$ and $G_{v \leftarrow D} \in SPG$.

Note that $C = A_{v \leftarrow C} \in SPG$ and $D = A_{v \leftarrow D} \in SPG$. We say that $v \leftarrow C$ and $v \leftarrow D$ are *elementary superpositions*. An example in Figure 1.2 characterizes the process of finding the graph $G_{v \leftarrow E}$.

Theorem 1.1. If $G, H \in SPG$ and v is an internal node of G, then $G_{v \leftarrow H} \in SPG$ (the class of superpositional graphs is closed under superposition).

Proof. By assumption $H \in SPG$ and $G \in SPG$ can be generated from graph A by some sequence of elementary superpositions. To show that $G_{v \leftarrow H} \in SPG$ we

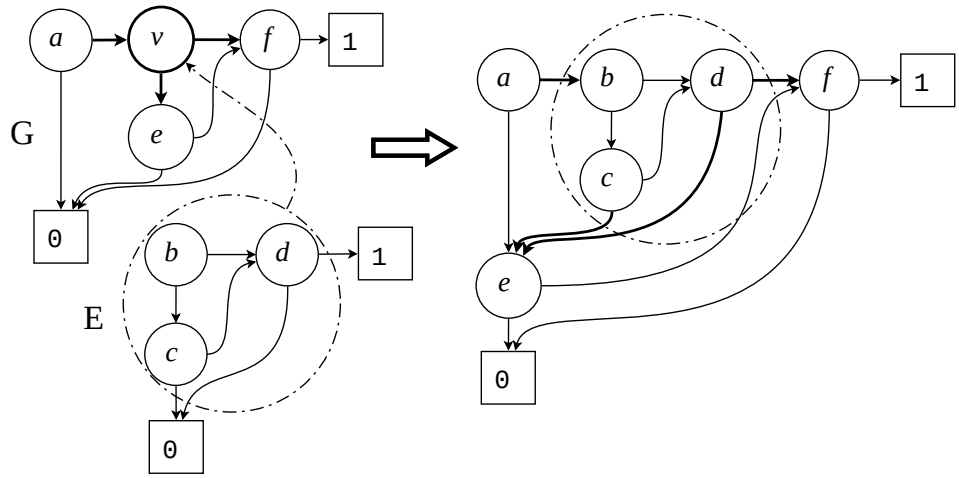


Figure 1.2: The superposition $v \leftarrow E$ in the graph G.

generate graph G using elementary superpositions and perform the same sequence of elementary superpositions in internal node v, which were implemented to get the graph H from the initial graph A. It is easy to see that the resulting graph $G_{v \leftarrow H}$ is in fact a superpositional graph. □

Theorem 1.1 allows us to give an alternative definition of the superpositional graphs, which is more convenient to use in the following proofs. Hereafter, if we say that $G \in SPG$ has n nodes, then terminal nodes 0 and 1 are not included in this count. In the figures we direct 1-edges from left to right and 0-edges from up to down, without the labels 1 and 0.

Definition 1.3. A *path* from node u to node v ($u \rightsquigarrow v$) is a sequence of nodes $w_0, \ldots, w_k$, where $w_0 = u$, $w_k = v$ and for each $0 \leq i < k$, $w_{i+1} = high(w_i)$ or $w_{i+1} = low(w_i)$.

Definition 1.4. A 0-*path* (1-*path*) from node u to node v is a path, which contains only 0-edges (1-edges).

Definition 1.5. A binary graph G is *traceable* if there exists a directed path through all internal nodes of G (Hamiltonian path).

A binary graph is acyclic, therefore, if the Hamiltonian path exists, then it is unique.

Definition 1.6. A binary graph G is *homogenous* if only one type of edges enter into every node $v \in V(G)$.

In the next theorem we list common properties of the class SPG, part of which are described in [4].

Theorem 1.2. Let $G \in SPG$. Then:

1. G has a unique root;
2. G is planar;
3. G is acyclic;
4. G is traceable;
5. for every internal node $v \in V(G)$ there exists a 0-path $v \rightsquigarrow 0$ and a 1-path $v \rightsquigarrow 1$;
6. G is homogenous.

Proof. We prove all these properties using induction over the structure of superpositional graphs, described by the alternative definition – Theorem 1.1. It is obvious that for graphs A, C and D all the properties hold. For the induction hypothesis let $G, E \in SPG$.

1. Let $S = G_{v \leftarrow E}$. If v is not the root of the graph G, then the root of the graph $G_{v \leftarrow E}$ is the node that was the root before this superposition. By induction hypothesis, root of the graph G is unique. If v is the root of G, then the root of $G_{v \leftarrow E}$ is the root of E but the root of E is unique by induction hypothesis.
2. Let $S = G_{v \leftarrow E}$. By induction hypothesis, graphs G and E are planar. By the definition of superposition, we can draw redirected edges in such a way that they do not intersect with the edges of the graph G nor with the edges of the graph E.
3. Let $S = G_{v \leftarrow E}$. By induction hypothesis, graphs G and E are acyclic. Since the graph E has only 2 sinks ($low(v)$ and $high(v)$) and there do not exist paths $low(v) \rightsquigarrow v$ and $high(v) \rightsquigarrow v$, it can be seen that the superposition does not produce any cycles.
4. Let $S = G_{v \leftarrow E}$. By induction hypothesis there exists an Hamiltonian path $u_1 \rightsquigarrow u_k$ in the graph G and the Hamiltonian path $w_1 \rightsquigarrow w_l$ in the graph E. The path $u_1 \rightsquigarrow u_k$ must include the node v, let $u_i = v$. By definition of superposition, in graph $S = G_{v \leftarrow E}$ we get $high(w_l) = high(u_i)$, $low(w_l) = low(u_i)$ and $w_1 = high(u_{i-1})$ or $w_1 = low(u_{i-1})$. Therefore, the Hamiltonian path in S is $u_1, \ldots, u_{i-1}, w_1, \ldots, w_l, u_{i+1}, \ldots, u_k$.
5. Let $S = G_{v \leftarrow E}$. By induction hypothesis there exists a 0-path $root(G) \rightsquigarrow 0$. If v does not belong to the path, then this 0-path remains unchanged in $G_{v \leftarrow E}$. If v is on the path $root(G) \rightsquigarrow 0$, we substitute v with the 0-path $root(E) \rightsquigarrow 0$, which exists according to induction hypothesis. The proof for 1-path is analogous.

6. Let $S = G_{v \leftarrow E}$. By induction hypothesis the graphs $G, E \in SPG$ are homogenous. By the execution of superposition $G_{v \leftarrow E}$, edges pointing to the node v were redirected to the root of E. Exiting 0-edges of the graph E were redirected to node $low(v)$; since $low(v)$ had at least one 0-edge in G, the graph remains homogeneous after substitution. The same argument works for $high(v)$ as well. □

Item 4 of Theorem 1.2 gives a canonical enumeration of the nodes of a superpositional graph. Given the canonical enumerations of $G, H \in SPG$, we can test the isomorphism between G and H in time $O(n)$ (we must check the endpoints of all edges and there are $2n$ edges in an n-node binary graph).

The finding of the Hamiltonian path of the binary graph G is a classical task of topological sorting nodes of the graph: find such in ordering

$$v_1 \prec v_2 \prec \ldots \prec v_n$$

of graph G where for every $v_i \rightarrow v_j$ it holds $i < j$. Due to the acyclicity of binary graphs, if the Hamiltonian path exists, then it is unique.

Definition 1.7. A *final node* of a binary graph G is a node, from which edges are pointing to terminals only.

If a binary graph G is traceable, then it has a unique final node – the last node v_n in its unique Hamiltonian path.

The idea of the following algorithm is to choose in every step such node, which has no successors. Thereafter, we remove the chosen node and redirect edges that are pointing to it.

Algorithm 1. Finding the canonical numeration of nodes of binarity graph.

find_Hamiltonian_path (binary graph G){
  $i := |V(G)|$;
  while ($i > 0$) {
    if (number of final nodes is not 1)
      return (0);
    $v :=$ final node;
    output v;
    redirect all edges pointing to node v to terminal nodes;
    $V(G) := V(G) \setminus \{v\}$;
    $i := i - 1$;
  }
}

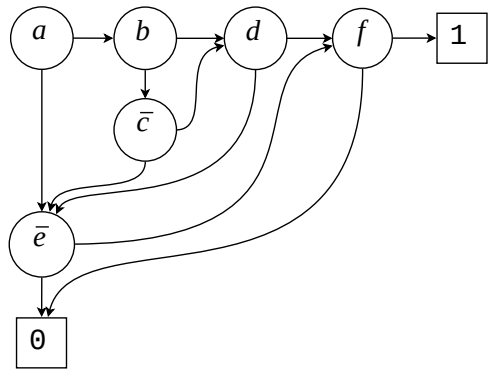


Figure 1.3: The SSBDD for the formula $a\&((((b \vee c)\&d) \vee e)\&f)$.

The output of Algorithm 1 is the Hamiltonian path of G in backward order. To find the following node of the Hamiltonian path, we must look at the nodes, where we redirected the exiting edges from. By using a data structure with backward pointers, the estimation of complexity is $\mathcal{O}(n)$.

1.3 SSBDD for Formulae

Let $\mathcal{F}$ be a propositional formula with variables $x_1, \ldots, x_n$, Boolean connectives $\&$, $\vee$, $\neg$, and parentheses. We suppose without loss of generality that all negations are attached directly to variables.

Definition 1.8. A *structurally synthesized binary decision diagram* $D(\mathcal{F})$ for a formula $\mathcal{F}$ is a superpositional graph, defined inductively according to the structure of $\mathcal{F}$ as follows:

1° if $\mathcal{F}$ is a literal l then $D(\mathcal{F})$ is a graph A, where the root is labelled by l;

2° if $\mathcal{F} = \mathcal{P}\&\mathcal{R}$ then $D(\mathcal{F})$ is the graph $C_{u \leftarrow D(\mathcal{P}), v \leftarrow D(\mathcal{R})}$;

3° if $\mathcal{F} = \mathcal{P} \vee \mathcal{R}$ then $D(\mathcal{F})$ is the graph $D_{u \leftarrow D(\mathcal{P}), v \leftarrow D(\mathcal{R})}$.

Binary graphs A, C, and D in Figure 1.1 are SSBDDs for formulae v, $u\&v$ and $u \vee v$ respectively. The formula

$$a\&\Big(\Big(\big((b \vee \overline{c})\&d\big) \vee \overline{e}\Big)\&f\Big)$$

has SSBDD, shown in Figure 1.3.

The notions of the activated path and the Boolean function, represented by the SSBDD, are similar to the case of BDD. The only difference is that, in order to choose the next element in the path, we have to evaluate literals instead of variables.

Theorem 1.3. A propositional formula $\mathcal{F}$ and its SSBDD $D(\mathcal{F})$ represent the same Boolean function.

Proof. We have to prove that for every $\alpha \in B^n$, $\alpha \vdash \mathcal{F}$ if and only if $\alpha \vdash D(\mathcal{F})$. We use induction by the definition of the formula $\mathcal{F}$. If $\mathcal{F} = l$ for some literal l and $\alpha \vdash l$, then $D(\mathcal{F})$ is the superpositional graph A, where $label(v)$ is l and the single path pointing to 1 in the graph A is activated by the assignment α, so $\alpha \vdash D(\mathcal{F})$. If $\alpha \nvdash l$, then the assignment α activates the path pointing to 0 in $D(\mathcal{F})$, so $\alpha \nvdash D(\mathcal{F})$.

If $\mathcal{F} = \mathcal{P}\&\mathcal{R}$ and $\alpha \vdash \mathcal{F}$, then $\alpha \vdash \mathcal{P}$ and $\alpha \vdash \mathcal{R}$. By induction hypothesis the assignment α activates paths pointing to 1 in $D(\mathcal{P})$ and $D(\mathcal{R})$. Then α activates the concatenation of these paths in $D(\mathcal{F})$ and $\alpha \vdash D(\mathcal{F})$. If $\alpha \vdash D(\mathcal{F})$, then by the construction of the $D(\mathcal{F})$, α activates paths pointing to 1 in $D(\mathcal{P})$ and $D(\mathcal{R})$. By induction hypothesis $\alpha \vdash \mathcal{P}$ and $\alpha \vdash \mathcal{R}$. Therefore $\alpha \vdash \mathcal{F}$.

If $\mathcal{F} = \mathcal{P} \vee \mathcal{R}$ and $\alpha \vdash \mathcal{F}$, then $\alpha \vdash \mathcal{P}$ or $\alpha \vdash \mathcal{R}$. Suppose $\alpha \vdash \mathcal{P}$. Then by induction hypothesis α activates some path pointing to 1 in $D(\mathcal{P})$, which is a path pointing to 1 in $D(\mathcal{F})$. Therefore $\alpha \vdash D(\mathcal{F})$. If $\alpha \nvdash \mathcal{P}$, then $\alpha \vdash \mathcal{R}$. By induction hypothesis α activates a path pointing to 0 in $D(\mathcal{P})$ and a path pointing to 1 in $D(\mathcal{R})$. Due to the construction of the SSBDD $D(\mathcal{F})$, α activates the concatenation of these paths in $D(\mathcal{F})$, which is a path pointing to 1. Therefore $\alpha \vdash D(\mathcal{F})$. If $\alpha \nvdash \mathcal{F}$, then by induction hypothesis α activates paths pointing to 0 in $D(\mathcal{P})$ and $D(\mathcal{R})$. The concatenation of these paths is a path pointing to 0 in $D(\mathcal{F})$, activated by α. Therefore $\alpha \nvdash D(\mathcal{F})$. □

Not every path of the $D(\mathcal{F})$'s superposition graph can be activated. Therefore, we define a *consistent* path of the SSBDD $D(\mathcal{F}(x_1, \ldots, x_n))$ as a path, which can be activated by some assignment $(\alpha_1, \ldots, \alpha_n) \in B^n$. The *satisfiability problem* for SSBDD can be formulated in the following way: does there exist a consistent path from the root to the sink 1? It is obvious that the number of intermediate nodes of the SSBDD $D(\mathcal{F})$ is the number of occurrences of variables in the formula $\mathcal{F}$. It follows that the satisfiability problem is $\mathcal{NP}$-complete and counting the number of true assignments is $\#\mathcal{P}$-complete for SSBDD representation of Boolean functions.

It is obvious that the conjunction of literals in a path from $root(F)$ to the terminal 1 in $D(\mathcal{F})$ is a term of the DNF for $\mathcal{F}$. Next theorem [16] says that we can remove from the terms of the DNF all literals, which are false for the corresponding assignment.

Definition 1.9. A *positive term* of the path $p = (v_1, \ldots, v_l)$ is the conjunction of literals of the nodes v_i, followed by the node $high(v_i)$:

$$\bigwedge_{\substack{1 \le i < l \\ v_{i+1} = high(v_i)}} label(v_i)$$

Theorem 1.4. Let $\mathcal{F}$ be a propositional formula and $D(\mathcal{F})$ its SSBDD. The disjunction of positive terms of all paths pointing to 1 in $D(\mathcal{F})$ is a DNF for the propositional formula $\mathcal{F}$.

Proof. Induction by the definition of the SSBDD. If $\mathcal{F}$ is a literal, say l, then $D(\mathcal{F})$ is the superpositional graph A, where l is an index of a single intermediate node v. A diagram $D(\mathcal{F})$ has one path pointing to 1: $(v, 1)$, where $high(v) = 1$ and $\mathrm{DNF}(A) = l$.

If $\mathcal{F}$ is the formula $\mathcal{F} = \mathcal{P}\&\mathcal{R}$, then $D(\mathcal{F})$ is the graph $C_{u \leftarrow D(\mathcal{P}), v \leftarrow D(\mathcal{R})}$. By induction hypothesis the theorem holds for diagrams $D(\mathcal{P})$ and $D(\mathcal{R})$. Every path pointing to 1 in $D(\mathcal{F})$ is the concatenation of some paths pointing to 1 in $D(\mathcal{P})$ and $D(\mathcal{R})$ and every positive term of a path pointing to 1 of $D(\mathcal{F})$ is the conjunction of some positive term of a path in $D(\mathcal{P})$ with some positive term of a path in $D(\mathcal{R})$. On the other hand, $\mathcal{F} \equiv \mathrm{DNF}(\mathcal{P})\&\mathrm{DNF}(\mathcal{R})$, so terms of $\mathrm{DNF}(\mathcal{F})$ are conjunctions of all terms of $\mathrm{DNF}(\mathcal{P})$ with all terms of $\mathrm{DNF}(\mathcal{R})$.

If $\mathcal{F}$ is the formula $\mathcal{F} = \mathcal{P} \vee \mathcal{R}$, then $D(\mathcal{F})$ is the graph $D_{u \leftarrow D(\mathcal{P}), v \leftarrow D(\mathcal{R})}$. By induction hypothesis the theorem holds for diagrams $D(\mathcal{P})$ and $D(\mathcal{R})$. Every path pointing to 1 in $D(\mathcal{F})$ is a path pointing to 1 in $D(\mathcal{P})$ or a concatenation of some path pointing to 0 in $D(\mathcal{P})$ with some path pointing to 1 in $D(\mathcal{R})$. From the concatenations there are essential ones only concatenation of 0-path from the root of the $\mathcal{D}(P)$ to the sink 0 with 1-path of $\mathcal{D}(F)$, because the positive term, corresponding to said path subsumes all other positive terms. On the other hand, $\mathcal{F} \equiv \mathrm{DNF}(\mathcal{P}) \vee \mathrm{DNF}(\mathcal{R})$, so $\mathrm{DNF}(\mathcal{F})$ consists of all the terms of both $\mathrm{DNF}(\mathcal{P})$ and $\mathrm{DNF}(\mathcal{R})$. □

1.4 SSBDD for Circuits

Let us represent the digital circuit by a set of equivalent parenthesis forms (EPF) synthesized by the superposition procedure directly from the circuit. At first we split the circuit into a set of tree-like (fanout-free) subcircuits by cutting all lines, which correspond to fanout > 1. To synthesize the EPF for a given tree, letters are first assigned to the gates and the lines. Then, starting from the output and working back towards primary inputs, EPF replaces individual literals by products of literals or sums of literals. When an AND gate is encountered during backtracing, a product term is created, in which the literals are the subscripted names of lines connected to the inputs of the AND gate. Encountering an OR gate causes a sum of literals to be formed, while encountering an inverter causes a literal to be complemented.

As an example, this procedure is illustrated to transform the subcircuit surrounded by the dotted line in left part of the Figure 1.4 to its EPF:

$$y = \overline{c_y e_y} = \overline{c_y} \vee \overline{e_y} = x_{6,c,y} x_{7_3,c,y} \vee d_{e,y} b_{e,y} =$$

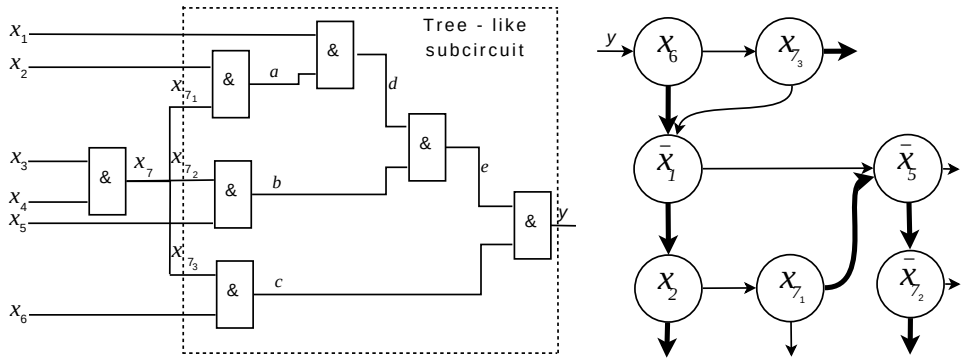


Figure 1.4: Logic circuit and an SSBDD for its tree-like subcircuit.

$$= x_{6,c,y}x_{7_3,c,y} \vee (\overline{x_{1,d,e,y}} \vee \overline{a_{d,e,y}})(\overline{x_{5,b,e,y}} \vee \overline{x_{7_2,b,e,y}}) =$$

$$= x_{6,c,y}x_{7_3,c,y} \vee (\overline{x_{1,d,e,y}} \vee x_{2,a,d,e,y}x_{7_1,a,d,e,y})(\overline{x_{5,b,e,y}} \vee \overline{x_{7_2,b,e,y}}) =$$

$$= x_6x_{7_3} \vee (\overline{x_1} \vee x_2x_{7_1})(\overline{x_5} \vee \overline{x_{7_2}})$$

In the circuit, the input variables x_k are substituted by indexes k for simplicity. In the case of fanout we use two indexes k_j for denoting the branches, where j numerates the branches. When creating the equation by the superposition procedure, the identity of every signal path from the inputs to the output of the given circuit is retained. Each literal in an EPF consists of a subscripted input variable or its complement, which identifies a path from the variable to the output. From the manner in which the EPF is constructed, it can be seen that there will be at least one subscripted literal for every path from each input variable to the output. The complemented literals correspond to paths, which contain an odd number of inversions. For simplicity, in the final EPF the list of indexes at literals are substituted by only the index of the input variable. Whole circuit is represented by the set of EPF-s, generated from trees. Every EPF is a propositional formula, whose variables are inputs of the circuit and new inputs, created by cutting fanouts.

For the next step we build an SSBDD for every EPF according to the definition in Section 1.3. An example of an SSBDD for the tree-like subcircuit is presented in Figure 1.4 right. For simplicity, the values of variables on edges in the SSBDD are omitted (by convention, the rightward edge always corresponds to 1 and the downward edge to 0). Also, the 0 and 1 sinks are omitted: leaving the SSBDD to the right always corresponds to $y = 1$, and down to $y = 0$. The set of SSBDDs for all tree-like (fanout-free) circuit segments represents the initial circuit.

1.5 Fault Simulation Using SSBDDs

In this section we will show how SSBDD models can be used for efficient modeling of faults in logic circuits [5]. Let us consider the *Single Stuck-At* (SSA) fault model, which is by far the most popular mathematical abstraction for modeling manufacturing defects in digital circuits. A fault in the SSA model is characterized by two properties:

1. only one circuit line is faulty:
2. the faulty line is permanently stuck to 0 or 1.

In practical applications of digital circuit test, one of the most common tasks is the fault simulation procedure. Fault simulation refers to checking whether a fault on a line is causing erroneous values in any of the circuit outputs under a given input variable assignment. Fault simulation for circuit inputs is related to Boolean derivative introduced in Section 1.1 as follows. Let F be the Boolean function representing the circuit and c_i be the circuit line corresponding to variable x_i. Fault simulation for a fault "c_i stuck-at $\neg\alpha_i$" will be equivalent to calculating the value of the Boolean derivative $\frac{\partial F}{\partial x_i}$ with a given assignment vector $(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n)$. If $\frac{\partial F}{\partial x_i}(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) = 1$ we say that vector $(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n)$ *covers* the fault "c_i stuck-at $\neg\alpha_i$".

As it can be seen from the EPF example shown in the previous section, each EPF literal and thus, each SSBDD node corresponds to exactly one structural path through the tree-like circuit segment. Consider the SSBDD in Figure 1.4, which represents the tree-like subcircuit. There are seven different paths through the subcircuit and thus, seven nodes in the corresponding SSBDD.

Fault simulation for SSBDD nodes takes place as follows. We say that the fault "stuck-at $\neg\alpha_i$" in node v_j, where $label(v_j)$ is x_i, is *covered* by the vector $(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n)$ iff all of the following three conditions are satisfied:

1. there exists an activated path from the root node to v_j;
2. there exists an activated path from $high(v_j)$ to the sink 1;
3. there exists an activated path from $low(v_j)$ to the sink 0.

Let us consider the example in Figure 1.4 with the assignment vector $\alpha = (1, 0, 0, 1, 0, 0)$ (i.e. $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, $x_4 = 1$, $x_5 = 0$, $x_6 = 0$). The paths activated in SSBDD by this vector are highlighted using bold arrows in Figure 1.4. For example, according to the above-mentioned criteria, fault "stuck-at 1" in the node v_1 labeled by x_6 is covered by α because all the three conditions are fulfilled. First, there exists an activated path from root (v_1 is the root node!). Second, there exists an activated path from $v_2 = high(v_1)$ to the sink 1. Finally, the path from $v_3 = low(v_1)$ via v_4 is activated by vector α. However, no faults in node v_3 are

Table 1.1: Number of collapsed and SSBDD faults in ISCAS'85 circuits.

circuit	uncollapsed	collapsed	SSBDD
c880	1550	**942**	994
c1355	2194	**1574**	1618
c1908	2788	1879	**1732**
c2670	4150	2747	**2626**
c3540	5568	3428	**3296**
c5315	8638	**5350**	5424
c6288	9728	**7744**	**7744**
c7552	11590	7550	**7104**

covered by vector α, due to the fact that condition 2 is not satisfied. Activated path from $v_6 = high(v_3)$ leads via v_6 to the sink 0 (instead of the sink 1!).

In practice, fault simulation for SSA faults is rarely carried out in all circuit lines. The list of faults considered is usually reduced to speed-up the simulation process. Such reduction is referred to as *fault collapsing*. There are several approaches to fault collapsing but they mainly rely on two kinds of relationships between faults: equivalence and dominance. We can consider modeling of SSA faults at SSBDD nodes as a special case of fault dominance. Faults at node v are dominated by the respective faults of the lines along the structural path corresponding to v. If a set of vectors covers all the faults at the SSBDD nodes, then all the faults at the corresponding subcircuit will also be covered.

Consider the Table 1.1, which compares the SSBDD based fault collapsing with the standard fault collapsing of ISCAS'85 benchmark circuits. In the Table 1.1, the second column shows the full uncollapsed fault numbers for the benchmarks, the third shows the standard collapsing fault numbers and the fourth column is for SSBDD fault count. The best results are denoted by bold digits.

However, it is not only the fault collapsing that makes SSBDD a favourable representation for fault modeling. Below are listed some of the special properties [16] of SSBDDs that can be used in further speeding up the fault simulation process.

Property 1. Let $p(\alpha) = p_1, \ldots, p_k$ be the activated path from the root node v_1 to the sink 1 (0, resp.). Then α can cover SSA faults only in these nodes $v_i, i = 1, \ldots, k$ along the path, whose labels $x_j = label(v_i)$ have assignments $\alpha_j = 1$ ($\alpha_j = 0$). Let us call such node faults *potential SSA faults*. Faults in other nodes of the SSBDD should not be simulated by vector α.

Property 2. Theorem 1.2 claimed that in a superpositional graph there always exists a directed path through all intermediate nodes. Let us assume that the nodes

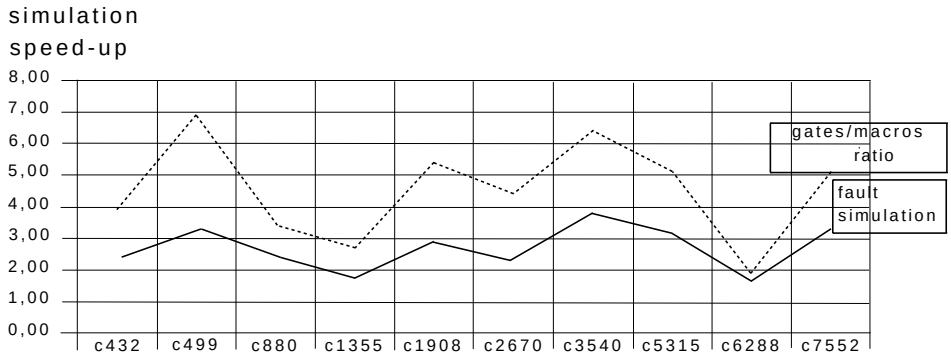


Figure 1.5: Comparison between fault simulation speed up and gates/subcircuits ratio.

in an SSBDD D are ordered according to this path. Let f^D be the Boolean function corresponding to the circuit represented by SSBDD D. If $f^D = 0$ ($f^D = 1$, resp.) at assignment α, if v_i is a node with a potential fault, and if an activated path from $high(v_i)$ ($low(v_i)$) crosses the activated path in the node v_k, then faults in $v_j, i \leq j < k$ are not covered by the vector α.

Based on these properties, an efficient fault simulation algorithm has been implemented and experiments on ISCAS'85 benchmark circuits have been carried out. The results of the experiments [5] are presented in Figure 1.5. As can be seen, the achieved speed-up ranges from 2 times (with c6288 circuit) to 7 times (c499) in comparison to traditional gate level fault simulation. The experiments also show that the effect is in good correlation with the ratio of the number of logic gates to the number of tree-like subcircuits for the given set of benchmarks.

CHAPTER 2

FINDING THE DESCRIPTION OF STRUCTURE BY COUNTING METHOD

The article [4], which the previous chapter was based on, was the first attempt to separate the properties of the underlying "skeleton" graphs from the properties of SSBDDs. Nevertheless, several problems remain unanswered:

1. whether a binary graph G is an SPG;
2. whether we can find some sequence of superpositions for $G \in SPG$ that generates it.

To determine whether the underlying graph is superpositional, we can try to generate these graphs according to the Definition 1.2. This means that we check all possible sequences of elementary superpositions of length $n-1$, where n is the number of nodes of the binary graph. This method is obviously infeasible.

We will start with a more general problem: which are the necessary and sufficient conditions for binary graphs to be SPGs. Since an SPG does not embody the history of its development (the sequence of superpositions that was used for generating the graph), then the desired properties cannot use the superposition. If we succeed in finding the necessary and sufficient conditions without superposition, then a better knowledge of SPGs will hopefully help us to find quick algorithms for solving to the two above-mentioned problems.

The methodology described in this chapter can be used, when the exact set of properties that describes the given structure cannot be found by "pure thought".

2.1 The Characteristic Integer Sequence for the Class SPG

The properties, proved in Theorem 1.2, are not sufficient to describe the class SPG . There exists a 4-node binary graph (Figure 2.1), which has all these properties but is not SPG. This graph becomes planar, if we draw the edge $v \to 0$ in a way that it does not cross edges $w \to z$ and $z \to 1$. Nevertheless, we cannot find a sequence of elementary superpositions generating it. Hence, some unlisted properties are still missing.

To find similar structures, we try to count n-node SPGs. By counting manually we find that the number of 1-node SPGs is 1, 2-node 2, 3-node 6, 4-node 22 and 5-node 90. The derived sequence $1, 2, 6, 22, 90, \ldots$ is long enough to check whether there exists a structure that describes the same sequence. Sequences like this and references to similar structures can be found in The Encyclopedia of Integer Sequences [11] and in its web version [10]. It turns out that the structure of the class SPG is very closely related to the Super-Catalan numbers (the Bracketing Problem, [13]) and the sequence matches with large Schröder numbers (Sequence $A006318$ in [10]). According to [10], the large Schröder numbers are twice Super-Catalan numbers except for the first term.

The Schröder Numbers [22]. The Schröder number S_n is the number of lattice paths in the Cartesian plane that start at $(0, 0)$, end at (n, n), contain no points above the line $y = x$, and are composed only of steps $(0, 1)$, $(1, 0)$, and $(1, 1)$, i.e., $\rightarrow$, $\uparrow$, and $\nearrow$. The diagrams illustrating the paths generating S_1, S_2, and S_3 are depicted in Figure 2.2.

The Bracketing Problem [13]. How many possibilities are there for placing brackets into n-length string so that:

- the initial string consists of natural numbers $1, 2, \ldots, n$;
- every pair of brackets contains at least 2 arguments;

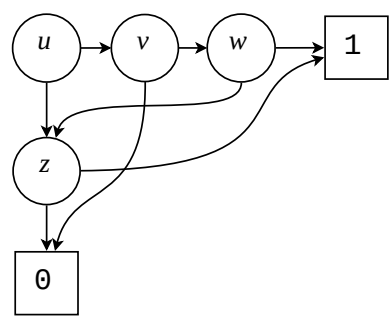


Figure 2.1: A graph satisfying the properties from Theorem 1.2, but is not SPG.

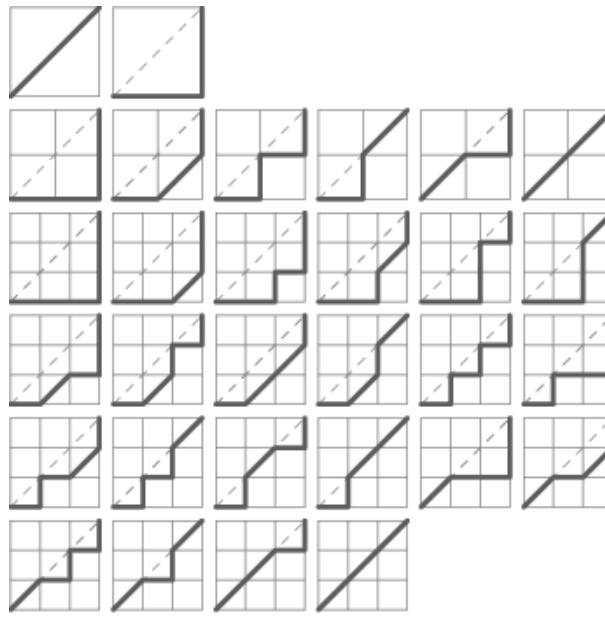


Figure 2.2: The Schröder paths pointing to the node $(1,1)$, $(2,2)$, and $(3,3)$.

- there are no brackets surrounding the whole string;
- for every left bracket there exists a right bracket and vice versa.

We say that such string with brackets is the *bracketing*. For example, in case $n = 4$ the number of bracketings is 11: 1234, $(12)34$, $1(23)4$, $12(34)$, $(123)4$, $1(234)$, $((12)3)4$, $(1(23))4$, $(12)(34)$, $1((23)4)$, and $1(2(34))$.

To prove that the detected similarity is not accidental, we need the follows definition.

Definition 2.1. A superpositional graph is a 0-*graph* if its Hamiltonian path consists of 0-edges only.

Definition 2.2. A superpositional graph is a 1-*graph* if its Hamiltonian path consists of 1-edges only.

Definition 2.3. A superpositional graph is a C-*graph* if it is

- a 1-graph or
- derived from the k-node ($k \geq 2$) 1-graph G by execution of the superposition

$$G_{v_1 \leftarrow G_1}, \ldots, G_{v_k \leftarrow G_k}$$

where G_i are D-graphs.

Definition 2.4. A superpositional graph is a *D-graph* if it is

- a 0-graph or
- derived from the k-node ($k \geq 2$) 0-graph G by execution of the superposition

$$G_{v_1 \leftarrow G_1}, \ldots, G_{v_k \leftarrow G_k}$$

where G_i are C-graphs.

Now it is obvious that if an SPG has $n \geq 2$ nodes, then it is either a C-graph or a D-graph but not both.

Since in the proof of the following theorem, the labels of the nodes are of high importance, we denote a k-node 0-graph by $L_0(l_1, \ldots, l_k)$ and a k-node 1-graph by $L_1(l_1, \ldots, l_k)$, where $l_1, \ldots, l_k$ are the labels of the nodes.

Theorem 2.1. In case $n = 1$, the number of bracketings is equal to the number of superpositional graphs. In case $n > 1$, exactly two superpositional graphs correspond to one bracketing.

Proof. If $n = 1$, then it is possible to place 0 pairs of brackets into the string 1. This corresponds to the graph A.

Now consider the case $n > 1$. We will show that the number of bracketings of an n-length string is precisely the number of n-node D-graphs.

We divide the bracketing $s_1 \cdots s_k$ - by pairs of brackets of external level – to substrings $s_1, \ldots, s_k$, where s_i is a natural number or a string in the form (s), where s is a bracketing. Construct the SPG by means of the following algorithm.

Input: bracketing $s_1 \cdots s_k$, where $k > 1$.

Output: D-graph G.

S0. $G := L_0(s_1, \ldots, s_k)$, where s_i is a natural number or string in the form (s).

Repeat the following steps S1 and S2 alternately until all labels of the nodes of the graph G are natural numbers.

S1. Apply $G := G_{(p_1 \cdots p_l) \leftarrow L_1(p_1, \ldots, p_l)}$ to every node of the graph G, whose label is in the form of $(p_1 \cdots p_l)$, where $l > 1$.

S2. Apply $G := G_{(p_1 \cdots p_l) \leftarrow L_0(p_1, \ldots, p_l)}$ to every node of the graph G, whose label is in the form of $(p_1 \cdots p_l)$, where $l > 1$.

An example in Figure 2.3 illustrates the process of finding the output in case of input bracketing $1((23)(45)6)7$.

Since it is uniquely determined in the process of generating the D-graph, from which 0-graph it is derived, and the 0-graph uniquely determines the outer brackets,

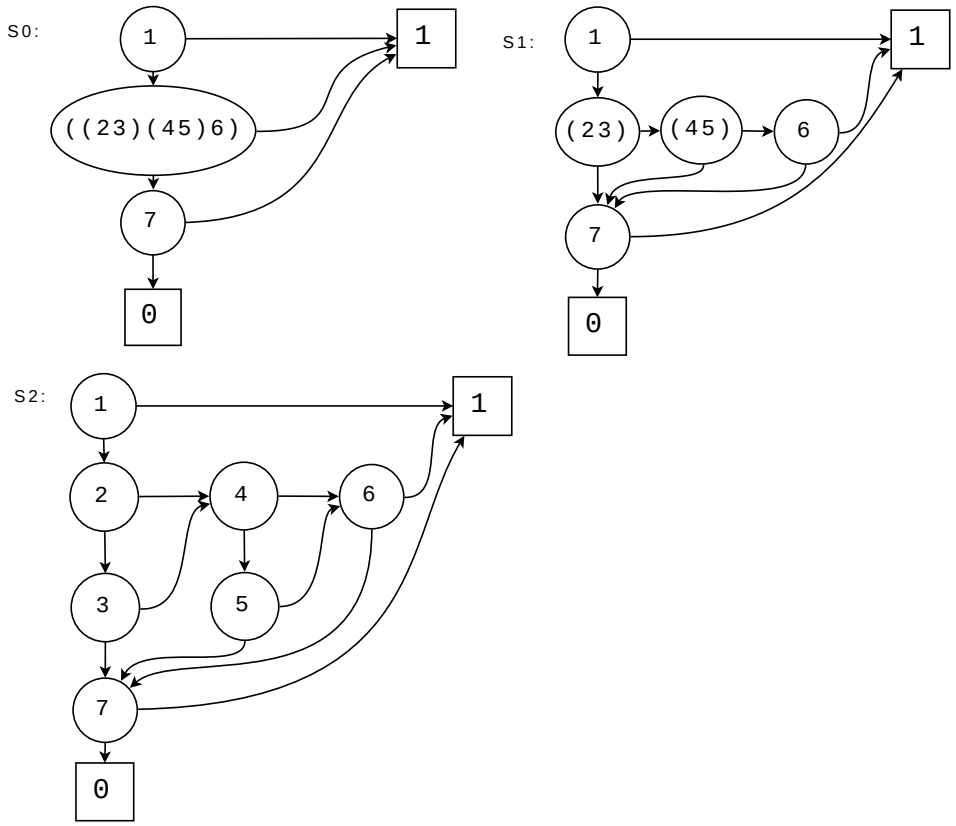


Figure 2.3: Application of the D-graph construction algorithm to the bracketing $1((23)(45)6)7$.

the correspondence between bracketings and D-graphs is bijective. Similarly, we can prove that the number of bracketings of an n-length string is precisely the same as the number of n-node C-graphs. Hence, the number of n-node SPGs is twice the number of bracketings of n-length string. □

Thus, as in [10], the characteristic integer sequence for the class SPG is

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, \ldots \tag{2.1}$$

This bijection is not very helpful – we could not translate any of the problems, described by large Schröder numbers, into the theory of superpositional graphs. We can use only the integer sequence as a starting point of our search for a solution to the problem.

2.2 Propositional Calculus in Combinatorics

Our next goal is to formulate a characteristic set of properties of the class SPG, knowing that the set must generate the Sequence 2.1. One possibility for doing this is by means of propositional formulae [12].

We try to find a family of propositional formulae $\mathcal{F}_n$, which depend on parameter n such that $\#SAT(\mathcal{F}_n)$ is the number of n-node superpositional graphs. Choosing the propositional variables in such a way that every satisfying assignment corresponds to some binary graph, we can interprete the formulae as the properties of binary graphs. Thereafter we must prove by traditional graph-theoretical methods that the set of properties found describes exactly the class SPG.

We illustrate the idea of describing the properties by means of propositional formulae by three examples.

2.2.1 Example I: Monotone Boolean Functions

Let $f\colon \{0,1\}^n \to \{0,1\}$ be a Boolean function of n variables. Define the relation $\alpha = \alpha_1 \dots \alpha_n \in \{0,1\}^n$ and $\beta = \beta_1 \dots \beta_n \in \{0,1\}^n$. Let

$$\alpha \prec \beta \equiv \exists i \left[\alpha_i < \beta_i \;\&\; \forall j \left((j \neq i) \to (\alpha_j = \beta_j)\right)\right].$$

By $\prec^+$ we denote the transitive closure of the relation $\prec$. The function $f\colon \{0,1\}^n \to \{0,1\}$ is *monotone*, if $f(\alpha_1, \dots, \alpha_n) \leq f(\beta_1, \dots, \beta_n)$, whenever $(\alpha_1, \dots, \alpha_n) \prec^+ (\beta_1, \dots, \beta_n)$.

Our goal is to describe the notion of monotonicity using a family of propositional formulae. Every Boolean function of n variables can be represented by its truth-table, so we use 2^n propositional variables $x_0, \dots, x_{2^n-1}$; one variable for each entry of the truth-table. For convenience, we suppose that the indexes of the variables are binary numbers. The correspondence of the Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ to the bit-string $f_0, \dots, f_{2^n-1}$ is defined by $f(\alpha) = f_\alpha$ for every $\alpha \in \{0,1\}^n$. Using this encoding, we can consider propositional formulae with variables $x_0, \dots, x_{2^n-1}$ as descriptions of the properties of n-variable Boolean functions [15].

It is easy to prove [14] that a function $f\colon \{0,1\}^n \to \{0,1\}$ with the truth-table $f_0, \dots, f_{2^n-1}$ is monotone if and only if the truth assignment $f_0, \dots, f_{2^n-1}$ satisfies the propositional formula

$$\mathcal{M}_n(x_0, \dots, x_{2^n-1}) \equiv \bigwedge_{\alpha,\beta \in \{0,1\}^n; \alpha \prec \beta} (\overline{x_\alpha} \vee x_\beta). \tag{2.2}$$

Formula (2.2) is a typical example of the description of the family of functions with one parameter n. By choosing the value of the parameter n we get a particular propositional formula. For $n = 3$ this formula is

$$\begin{gathered}\mathcal{M}_3(x_{000}, x_{001}, x_{010}, x_{100}, x_{011}, x_{101}, x_{110}, x_{111}) \equiv \\ (\overline{x}_{000} \vee x_{001})\&(\overline{x}_{000} \vee x_{010})\&(\overline{x}_{000} \vee x_{100})\&(\overline{x}_{001} \vee x_{011})\& \\ (\overline{x}_{001} \vee x_{101})\&(\overline{x}_{010} \vee x_{011})\&(\overline{x}_{010} \vee x_{110})\&(\overline{x}_{100} \vee x_{101})\& \\ (\overline{x}_{100} \vee x_{110})\&(\overline{x}_{011} \vee x_{111})\&(\overline{x}_{101} \vee x_{111})\&(\overline{x}_{110} \vee x_{111}). \qquad (2.3)\end{gathered}$$

Formula (2.3) is a propositional formula and it can be given to some program for satisfiability testing (which gives the answer YES because there exist monotone functions of three variables) or a program for counting solutions (which gives the answer 20 because there exist 20 monotone functions of three variables). One can also use a program for evaluating propositional formulae to test the monotonicity of any three-variable Boolean function given by its truth-table.

2.2.2 Example II: The Pigeonhole Problem

The pigeonhole problem [21] involves showing that it is impossible to put $n+1$ pigeons into n holes if each pigeon must go into a distinct hole. The n-hole ($n+1$ pigeon) problem can be described using $n^2 + n$ variables, where the variable $x_{i,j}$ ($1 \leq i \leq n$, $1 \leq j \leq n+1$) is interpreted as which of the i-th hole contains the j-th pigeon. The unsatisfiable formula consists of two parts:

- no two pigeons share a hole;
- each pigeon must be in a hole.

The actual family of formulae is

$$\mathcal{F} \equiv \left(\bigwedge_{\substack{1 \leq j,k \leq n+1; j \neq k \\ 1 \leq i \leq n}} (\overline{x}_{i,j} \vee \overline{x}_{i,k}) \right) \& \left(\bigwedge_{1 \leq j \leq n+1} \bigvee_{1 \leq i \leq n} x_{i,j} \right). \qquad (2.4)$$

2.2.3 Example III: Graph Planarity

In [12] Hermann Stamm-Wilbrandt deduced the family of propositional formulae expressing graph planarity, where the parameter is the graph $G = (V, E)$, using the fact that a graph is planar if and only if it has so-called horvert representation:

$$
\begin{aligned}
\mathcal{F} \quad \equiv \quad & \bigwedge_{v \in V(G)} exactlyone(x_{v,i} : 1 \le i \le |V(G)|) \\
& \& \bigwedge_{1 \le i \le |V(G)|} exactlyone(x_{v,i} : v \in V(G)) \\
& \& \bigwedge_{1 \le j \le |V(G)|} \bigwedge_{1 \le i < 2|V(G)|-4} \bigwedge_{i < h \le 2|V(G)|} (x_{i,j} \& \neg x_{i+1,j} \rightarrow \neg x_{h,j}) \\
& \& \bigwedge_{e \leftarrow (u,v) \in E(G)} atleastone(x_{e,i} : 1 \le i \le 2|V(G)| - 4) \\
& \& \bigwedge_{e \leftarrow (u,v) \in E(G)} \bigwedge_{1 \le i \le |V(G)|} \bigwedge_{1 \le j \le 2|V(G)|-4} (x_{u,i} \& x_{e,j} \rightarrow x_{j,i}) \\
& \& \bigwedge_{e \leftarrow (u,v) \in E(G)} \bigwedge_{1 \le j \le |V(G)|} \bigwedge_{1 \le i \le 2|V(G)|-4} (x_{v,j} \& x_{e,i} \rightarrow x_{i,j}) \\
& \& \bigwedge_{e \leftarrow (u,v) \in E(G)} \bigwedge_{1 \le i,j \le |V(G)|; i \ne j} \bigwedge_{1 \le h \le 2|V(G)|-4} \bigwedge_{\min(i,j) < k < \max(i,j)} \\
& (x_{u,i} \& x_{v,j} \& x_{e,h} \rightarrow \neg x_{h,k}) \qquad (2.5)
\end{aligned}
$$

For every particular graph G we can transform (2.5) to a propositional formula, which is satisfiable if and only if G is planar.

2.3 Methodology of Finding the Description of Structure by Counting Method

The Sequence (2.1) really counts the numbers of isomorphic classes of SPG. Our next goal is to find such a family of propositional formulae $\mathcal{F}_n$, which would depend on parameter n (the number of nodes in SPG), and, when valuating integers $1, 2, 3, \ldots$ to the parameter n, would produce the Sequence (2.1) as the answer to the counting problem $\#SAT(\mathcal{F}_n)$.

Let us suppose $\mathcal{F}_n^*$ is the required propositional formula. When searching for the formula and getting some approximation $\mathcal{F}_n$ for it, it might happen that

$$\#SAT(\mathcal{F}_n^*) = \#SAT(\mathcal{F}_n)$$

holds if $n < k$, but does not hold if $n = k$. There are two possibilities:

- $\#SAT(\mathcal{F}_k^*) < \#SAT(\mathcal{F}_k)$, the described set of properties is underconstrained. Try to find some extra property or constrain some of them;
- $\#SAT(\mathcal{F}_k^*) > \#SAT(\mathcal{F}_k)$, the set of properties found is overconstrained.

In the first case, there exists an assignment α, where $\mathcal{F}_k(\alpha) = true$, but the assignment α does not represent any SPG. In the second case, there exists the assignment β that represents a certain SPG, but $\mathcal{F}_k(\beta) = false$. An analysis of the received problematical assignments (graphs) provides us with information for improving the properties. The goal is to refine the logical description until the received integer sequence coincides with the Sequence (2.1). The last step is to prove by traditonal graph-theoretical methods that the set of properties found describes exactly the class SPG.

In addition, the integer sequences that form the basis of the methodology are not necessarily in the Encyclopedia of Integer Sequences, but could also be found by counting by brute force.

CHAPTER 3

THE LANGUAGE AND THE TRANSLATOR FOR FAMILIES OF PROPOSITIONAL FORMULAE

Our goal is to find the description of the class SPG by means of the counting method, described in Section 2.3. A bottleneck of this method could be the fact that we must calculate manually or write a separate program for each of the found approximations, which constructs a *classical* propositional formula for every value of the family parameter.

The problem will be solved by the translator for metaformulae [8, 25], which translates the description of a parameterized family of propositional formulae to propositional formula corresponding to the value of the parameter. In the present section we describe the grammar of LaTeX-format propositional formulae, accepted as the inputs of the translator, and variations of it and also explain how the above-mentioned translator works [8, 25].

The translator is implemented by using a universal parser for mixed strategy precedence grammars. The analyzer transforms the input text into a semantic tree and a compiler performs the transformation of the tree to the output.

3.1 Description of Input and Output

We do not have much freedom to choose the format for input and output languages. The examples in Section 2.2 show that the description of the family of propositional formulae can be very sophisticated. So we use LaTeX source for input.

There exists a common standard format – DIMACS standard [23] – which is supposed to be the standard representation of propositional formulae for every program in the toolkit of propositional logic. This is the reason why the first output of the translator is DIMACS `satex` format.

The second output format is `tav`. The `tav`-format file contains a propositional formula in the prefix form, in which, apart from classical operators and, or, neg and $impl$, we can use operators odd, $even$, $exactlyone$, $atmostone$, $atleastone$, $none$ and eq:

$$
\begin{aligned}
odd(x_i : 1 \le i \le n) &\equiv \bigoplus_{1\le i\le n} x_i \\
even(x_i : 1 \le i \le n) &\equiv \neg \bigoplus_{1\le i\le n} x_i \\
atleastone(x_i : 1 \le i \le n) &\equiv \bigvee_{1\le i\le n} x_i \\
atmostone(x_i : 1 \le i \le n) &\equiv \bigwedge_{1\le i,j\le n; i<j} (\overline{x}_i \vee \overline{x}_j) \\
exactlyone(x_i : 1 \le i \le n) &\equiv \Big(\bigvee_{1\le i\le n} x_i\Big) \& \Big(\bigwedge_{1\le i,j\le n; i<j} (\overline{x}_i \vee \overline{x}_j)\Big) \\
none(x_i : 1 \le i \le n) &\equiv \neg \bigvee_{1\le i\le n} x_i \\
eq(x_i : 1 \le i \le n) &\equiv \Big(\bigwedge_{1\le i\le n} x_i\Big) \vee \Big(\bigwedge_{1\le i\le n} \overline{x}_i\Big)
\end{aligned}
$$

The `tav`-format also allows more effective solution to the problems of satisfiability checking and counting.

3.1.1 Input Language

Obviously, an arbitrary formula written as a LaTeX source can not be considered as a description of a family of propositional formulae. Therefore, we have to give the exact definition of the family. We use the Backus-Naur Form (BNF) as a syntactical formalism.

```
P1:  <family>        ::= $$<metaformula>$$
P2:  <metaformula>   ::= <input><bigformula>
P3:                  ::= <bigformula>
P4:  <input>         ::= INPUT(graph<graphnames>)
P5:                  ::= INPUT(var<varnames>)
P6:                  ::= INPUT(var<varnames>;
                         graph<graphnames>)
P7:  <graphnames>    ::= <graphnames>,<graphname>
P8:                  ::= <graphname>
P9:  <graphname>     ::= #i#
P10: <varnames>      ::= <varnames>,<variable>
```

```
P11:                    ::= <variable>
P12: <variable>         ::= #i#
```

The <input> part of the description (productions P4–P12) is the tool for establishing the correspondence of formal parameters with concrete values given in the command line. The value of the <graphname> must be the name of the file, which contains the description of the graph in DIMACS standard format (Section 8.2 in [24]). If a metaformula does not contain input commands, then the values of formal parameters are determined in the dialogue with the user.

The string #i# is a shorthand for the lexical class *identifier*. Below we use one more lexical class #c# for integer constants.

```
P13: <bigformula>       ::= \bigvee_{<limits>}<bigformula>
P14:                    ::= \bigwedge_{<limits>}
                            <bigformula>
P15:                    ::= \bigoplus_{<limits>}
                            <bigformula>
P16:                    ::= <prefixop>(<bigformula>:
                            <limits>)
P17:                    ::= <formula>
P18: <prefixop>         ::= odd
P19:                    ::= even
P20:                    ::= exactlyone
P21:                    ::= atmostone
P22:                    ::= atleastone
P23:                    ::= none
P24:                    ::= eq
```

The main syntactical unit in the description of the family of formulae is the <bigformula>. The syntax of LaTeX provides three large logical operators: \bigvee, \bigwedge and \bigoplus. We have augmented the input language with the productions P16–P26 to allow some useful constructions with similar interpretation (but different syntax). The exact semantics of prefix-operators is defined by the equations in Section 3.1.

```
P25: <limits>           ::= <limit>;<constraints>
P26:                    ::= <limit>
P27: <limit>            ::= <metavariables>\in<set>
P28:                    ::= <node>\in V(<graphname>)
P29:                    ::= <arc>\in E(<graphname>)
P30:                    ::= <expression><lessrel>
                            <metavariables><lessrel>
```

```
                          <expression>
P31: <metavariables> ::= <metavariables>,
                          <metavariable>
P32:                 ::= <metavariable>
P33: <metavariable>  ::= #i#
P34: <set>           ::= B^{<expression>}
P35: <node>          ::= <metavariable>
P36: <arc>           ::= <arcname>
P37:                 ::= <arcname>\leftarrow\{<node>,
                          <node>\}
P38:                 ::= \{<node>,<node>\}
P39: <arcname>       ::= <metavariable>
P40: <lessrel>       ::= <
P41:                 ::= \leq
P42:                 ::= \prec
P43:                 ::= \prec^+
P44: <constraints>   ::= <constraints>,<constraint>
P45:                 ::= <constraint>
P46: <constraint>    ::= <expression><relation>
                          <expression>
P47: <relation>      ::= <lessrel>
P48:                 ::= =
P49:                 ::= \neq
P50:                 ::= >
P51:                 ::= \geq
```

The idea of translating the description of the family into the propositional formula is unfold `<formula>` for every combination of `<metavariables>` allowed by `<limits>` and constrained by `<constraints>`. Boolean formula within the scope of `<bigvee>`, `<bigwedge>`, or `<bigoplus>` is supposed to have Boolean variables indexed by metavariables or arithmetical expressions.

There are four types of metavariables. Productions `P27` and `P34` define the type *bitvector*, here `B^n` means $\{0,1\}^n$. The productions `P28`, `P35` define a metavariable of type *graph node*. Productions `P29` and `P36`–`P39` define the type *graph arc*; one can use the name of the arc, names of endpoints of the arc or both afterwards. The production `P39` allows using *integer* variables.

```
P52: <expression>    ::= <expression>+<term>
P53:                 ::= <expression>-<term>
P54:                 ::= <term>
P55: <term>          ::= <factor>*<term>
P56:                 ::= <factor>/<term>
P57:                 ::= <factor>
```

```
P58: <factor>         ::= <factor>^<primary>
P59:                  ::= <primary>
P60: <primary>        ::= #c#
P61:                  ::= <metavariable>
P62:                  ::= |V(<graphname>)|
P63:                  ::= |E(<graphname>)|
P64:                  ::= |<metavariable>|
P65:                  ::= min\{<expression>,
                          <expression>\}
P66:                  ::= max\{<expression>,
                          <expression>\}
P67:                  ::= (<expression>)
```

In production P52, <expression> is a traditional arithmetical expression with operations $+$, $-$, $*$, /, ^, brackets and metavariables and constants as operands. In addition, there are three special functions that can be used in expressions: |V(<graphname>)|, |E(<graphname>)|, and |<metavariable>|. These give the number of nodes and arcs of the graph and the number of ones in the bit-string. The functions max and min have traditional semantics.

```
P68: <formula>        ::= <leftpart>\sim<formula>
P69:                  ::= <leftpart>\oplus<formula>
P70:                  ::= <leftpart>\to<formula>
P71:                  ::= <leftpart>
P72: <leftpart>       ::= <leftpart>\vee<clause>
P73:                  ::= <clause>
P74: <clause>         ::= <logterm>\&<clause>
P75:                  ::= <logterm>
P76: <logterm>        ::= \neg<literal>
P77:                  ::= <literal>
P78: <literal>        ::= <logvar>
P79:                  ::= <logvar>_{<indexes>}
P80:                  ::= <prefixformula>
P81:                  ::= (<bigformula>)
P82: <indexes>        ::= <index>,<indexes>
P83:                  ::= <index>
P84: <index>          ::= <expression>
P85: <prefixformula>  ::= <prefixop>(<formlist>)
P86: <formlist>       ::= <formlist><bigformula>
P87:                  ::= <bigformula>
P88: <logvar>         ::= #i#
```

A <formula> is a propositional formula, which is a mixture of infix and prefix style. Operands are propositional variables, which can be indexed by arith-

metical expressions of constants and meta-variables. In addition, negations have to be in front of literals (rules `P76`–`P81`).

Some deviations from the grammar are still allowed, especially if the aim is to achieve a better layout. Allowed deviations from the grammar are described in Section 3.2.

3.1.2 Output Language

The first output language is the DIMACS format `satex` (Sections 2.2 and 2.3 in [23]). The variables in DIMACS format are represented by numbers 1 through n. The formulae are in prefix form: $-f$, $*(f_1\ f_2\ \ldots\ f_k)$, $+(f_1\ f_2\ \ldots\ f_k)$, $xor(f_1\ f_2\ \ldots\ f_k)$ and $=(f_1\ f_2\ \ldots\ f_k)$, where $f, f_1, f_2, \ldots, f_k$ are formulae.

The second output language is the `tav`-format. The variables in the `tav`-format are represented by strings $x1$ through xn. The output file `<filename>.tav` contains one propositional formula in the prefix form in which the operators and, or, $impl$, neg, odd, $even$, $exactlyone$, $atmostone$, $atleastone$, $none$, and eq can be used. The arguments of these operators must be given in brackets, separated by commas. The arguments are propositional formulae or variables in the `tav`-format.

To better understand the formula, one has to know the encoding of the original variables. Therefore, the translator additionally outputs the table of correspondence of variables.

3.2 The Preprocessor of the Translator

The preprocessor of the translator allows the input formula to deviate to some extent from the strict rules of the translator. In addition, the preprocessor allows the formula to be built up from parts. This means that the user does not have to insert repeated parts of the formula separately; these parts can be described and then used an arbitrary number of times.

3.2.1 The Working Principle

The preprocessor gets the name of a text file in the LaTeX format as the input of user dialogue. It finds the parts of the formula in the text file(if there are any). The formula to be analysed must be the first mathematical expression between dollar signs or pairs of dollar signs that does not contain the equation of the type

```
<name of the function>_{<number>}=...
```

As far as the rest of the components is concerned, there are no constraints except one: the components have to be in the same file (and between dollar signs).

After the components of the formula have been read from the file, the correction of formula parts takes place (Section 3.2.2). After the correction of formula parts, the valuating of parameters and necessary replacements (implied by the composition of the formulae) are performed and the main formula is written to the output file `<filename>.frm`.

Example. The pigeonhole problem described in Section 2.2.2 has an equivalent representation

$$F_1 \& F_2,$$

where

$$F_1 = \bigwedge_{1\le j,k\le n+1} \bigwedge_{1\le i\le n; i\ne k} \Big(\overline{x}_{i,j} \vee \overline{x}_{i,k}\Big),$$

$$F_2 = \bigwedge_{1\le j\le n+1} \bigvee_{1\le i\le n} x_{i,j}.$$

The content of the input file corresponding to this is

```
$$F_{1}\&F_{2}$$

where

$$F_{1}=\bigwedge_{1\le j,k\le n+1}
        \bigwedge_{1\le i\le n; i\ne k}
        \Big(\overline x_{i,j}\vee\overline x_{i,k}\Big)$$

$$F_{2}=\bigwedge_{1\le j\le n+1}
        \bigvee_{1\le i\le n}x_{i,j}$$
```

The definition of every subformula F_i may also contain calls of other analogous subformulas.

3.2.2 Rules for Replacement, Allowed Deviations from Grammar

Next we present all the deviations from the grammar, allowed by the preprocessor. In the left column of the table below there are expression the grammar rules and the right column can replace them expressions, that in the source file. The preprocessor transforms all the expressions on the right to the corresponding expressions on the left; for example, `Odd` is corrected to `odd`.

```
K1:    "odd"              "Odd"
K2:    "even"             "Even"
K3:    "atleastone"       "Atleastone", "at_least_one",
```

```
                          "At_Least_One"
K4:    "exactlyone"       "Exactlyone", "exactly_one",
                          "Exactly_One"
K5:    "atmostone"        "Atmostone", "atmost_one",
                          "Atmost_One"
K6:    "atleast"          "Atleast", "at_least",
                          "At_Least"
K7:    "atmost"           "Atmost", "at_most", "At_Most"
K8:    "level"            "Level"
K9:    "treshold"         "Treshold"
K10:  "none"              "None"
K11:  "\to"               "\Rightarrow", "\rightarrow"
K12:  "\sim"              "\Leftrightarrow",
                          "\leftrightarrow"
K13:  "\&"                "\wedge", "\land"
K14:  "\vee"              "\lor"
K15:  "\}"                "\rbrace"
K16:  "\{"                "\lbrace"
K17:  "\leq"              "\le", "\leqslant"
K18:  "*"                 "\ast", "\cdot", "\times"
K19:  ""                  "\ "
K20:  ""                  "\enskip", "\quad","\qquad",
                          "\,", "\>", "\;", "\!"
K21:  ""                  "\indent", "\\", "\nonumber",
                          " & "
K22:  "\neg "             "\overline", \lnot""
K23:  ""                  " "
K24:  ""                  "&&", "\\", "\break"
K25:  "("                 "\big(", "\Big(", "\bigg(",
                          "\Bigg(", "\lgroup", "\left(
K26:  ")"                 "\big)", "\Big)", "\bigg)",
                          "\Bigg)", "\rgroup", "\right)"
K27:  "\}"                "\big\}", "\Big\}", "\bigg\}",
                          "\Bigg\}"
K28:  "\{"                "\big\{", "\Big\{", "\bigg\{",
                          "\Bigg\{"
K29:  ""                  "\limits"
K30:  "$$"                "\begin{displaymath}",
                          "\end{displaymath}"
K31:  "\ne"               "\neq"
K32:  "\geq"              "\ge"
K33:  "|"                 "\mid", "\vert", "\arrowvert",
                          "\bracevert"
K34:  "\leftarrow"        "\gets"
K35:  "x_{"               "[", "\lbrack"
K36:  "}"                 "]", "\rbrack"
K37:  "\sim"              "\thicksim"
K38:  ""                  "\eqno(...)", "\leqno(...)"
```

Rules K1–K14 add additional possibilities to write down logical operators. Rule K18 is for removing various multiplication signs. Rules K19, K20, K21, and K29 are for removing some types of indentitations, horizontal spaces and LaTeX formatting operators that are unnecessary at this point. Replacement K22 is necessary, because in some papers, (for example [12]), the sign \overline or \lnot is used instead of \neg to denote negation. We definitely remove excess spaces (K23) and tabulator and line signs (K24). Replacement of large brackets with normal size brackets is controlled by the rules K25–K28. Rule K30 unifies the delimiters in the formulae. The rest of the rules is necessary to replace the synonyms of the operators with the ones defined in the grammar.

One cannot use any other LaTeX operators than those that are given in rules K1–K38 and in the grammar for writing down formulae.

3.3 Running the Translator

The translator can be called using the command line

```
>metacomp <filename> [<parameters>] [<graphs>]
```

where <filename>.tex is the name of the file, containing the description of a family of propositional formulae, <parameters> are integer parameters of the family and <graphs> are filenames of graph parameters of the family. The order of parameters must correspond to the order of formal parameters in the <input> part of the description of the family. The result of the translator consists of three files: <filename>.sat contains the translated formula in DIMACS standard format, <filename>.tav contains the translated formula in tav-format, and <filename>.aux contains the table of correspondence of variables.

The translation starts with preprocessing phase (Section 3.2), which unifies all high level LaTeX constructs and removes all LaTeX formatting commands according to the replacement rules.

3.3.1 Example

As an example, we use the family (2.2) for describing monotone Boolean functions. Let the file monbool.frm be

```
\begin{equation}
INPUT(var n)
\bigwedge_{\alpha,\beta\in B^{n};\alpha\prec\beta}
\left(\overline{x_{\alpha}}\vee x_{\beta}\right)
\end{equation}
```

Apply the translator using the command line

```
>metacomp monbool 3
```

After preprocessing we have the text

```
#
INPUT(var n)
\bigwedge_{\alpha,\beta\in B^{n};\alpha\prec\beta}
(\neg x_{\alpha}\vee x_{\beta})
#
```

The output consists of three files. File `monbool.sat` is

```
c Family monbool.frm n=3
p sat 8
(*(+(-1 2)
   +(-1 3)
   +(-1 4)
   +(-2 5)
   +(-2 6)
   +(-3 5)
   +(-3 7)
   +(-4 6)
   +(-4 7)
   +(-5 8)
   +(-6 8)
   +(-7 8)))
```

File `monbool.tav` is

```
and(
or(not(x1),x2),or(not(x1),x3),
or(not(x2),x4),or(not(x3),x4),
or(not(x1),x5),or(not(x2),x6),
or(not(x5),x6),or(not(x3),x7),
or(not(x5),x7),or(not(x4),x8),
or(not(x6),x8),or(not(x7),x8)
)
```

File `monbool.aux` is

```
Table of correspondences
Family monbool.frm n=3
x(000)  = 1
x(001)  = 2
x(010)  = 3
x(100)  = 4
x(011)  = 5
x(101)  = 6
x(110)  = 7
x(111)  = 8
```

CHAPTER 4

DESCRIPTION OF SUPERPOSITIONAL GRAPHS WITHOUT TERMINAL NODES

In this chapter we present the first description that we found for the class SPG and describe the process of finding it. To find the description, we start with the selection of variables.

When drawing SPGs, which form the basis for describing SSBDD, we always point 0-edges from up to down and 1-edges from left to right. Therefore, it is a tradition ([4], Figure 1.3 in this thesis) to omit the edges pointing to terminal nodes and draw them only when there is a clear need for it. It is obvious that if there is a 0-edge or a 1-edge missing at some node, then according to the previously mentioned rule, it is unambiguously retrievable, which terminal node this edge is pointing to.

Next we describe the set of propositional variables for a binary graph and try to find a propositional formula describing the class SPG.

4.1 Representation of Binary Graphs by Propositional Variables

According to the Theorem 1.2, there exists a unique Hamiltonian path in $G \in SPG$, which gives a canonical enumeration of the nodes of G. Let the Hamiltonian path of an n-node SPG G consist of nodes $v_1, \ldots, v_n$.

The meaning of the propositional variables $x_{i,j}$ and $y_{i,j}$ is, respectively, the existence of 0-edge and 1-edge from node v_i to node v_j:

$$x_{i,j} = \begin{cases} 1, & \text{if there exists a 0-edge } v_i \to v_j; \\ 0, & \text{otherwise;} \end{cases}$$

$$y_{i,j} = \begin{cases} 1, & \text{if there exists a 1-edge } v_i \to v_j; \\ 0, & \text{otherwise.} \end{cases}$$

Since SPGs are acyclic, then in case there exists an edge from node v_i to node v_j, certainly $i < j$. Therefore, propositional formulae for the class SPG are defined on the set

$$X = \{x_{i,j}, y_{i,j} : 1 \leq i < j \leq n\}$$

and the required formula $\mathcal{F}_n^*(X)$ should satisfy the condition

$$\mathcal{F}_n^*(\alpha) = \begin{cases} 1, & \text{if the graph represented by } \alpha \text{ is an SPG;} \\ 0, & \text{otherwise;} \end{cases}$$

for every assignment α to X.

4.2 Descriptive Properties of the Class SPG as Propositional Formulae

The properties of the class SPG described in Sect. 1.2 now be expressed by means of propositional formulae. In the descriptions, we use the operators and, or, xor, $\to$ (implication), $\neg$ (negation) and $exactlyone$ (allowed operators and rules can be found in Chapter 3).

In each successive approximation, we add a new property or remove any excessive ones. In order to see whether a property is unnecessary or not, we check each of the resulting approximations with the counter of satisfying assignments. In addition, the set of properties that has already been found (see Theorem 1.2) needs to be checked to see whether any of the properties are unnecessary. Thus we select a set with the smallest number of properties for each approximation.

Approximation I: formulas for known properties. The fact that the graph is acyclic is already taken into account when selecting the variables.

There is exactly one edge (0-edge or 1-edge) between successive nodes of Hamiltonian path:

$$\mathcal{F}_1 \equiv \bigwedge_{1 \leq i \leq n-1} xor(x_{i,i+1}, y_{i,i+1}).$$

The graph is binary, i.e., at most one 0-edge exits from each node:

$$\mathcal{F}_2 \equiv \bigwedge_{1 \leq i \leq n-1} atmostone(x_{i,j} : i < j \leq n).$$

Also there is at most one 1-edge exiting from each node:

$$\mathcal{F}_3 \equiv \bigwedge_{1 \leq i \leq n-1} atmostone(y_{i,j} : i < j \leq n).$$

Since the described properties must hold for every SPG, an approximation of the formula $\mathcal{F}_n^*$ for the class SPG is

$$\mathcal{F} \equiv \mathcal{F}_1 \& \mathcal{F}_2 \& \mathcal{F}_3.$$

After applying the translator of propositional formulae to the formula $\mathcal{F}$ and then the counter of satisfying assignments to the result, we get the sequence

$$1, 2, 8, 48, 384, 3840, 46080, \ldots$$

With the use of the translator and the counter, we convince ourselves that the corresponding terms of sequences of formulae $\mathcal{F}_1 \& \mathcal{F}_2$, $\mathcal{F}_2 \& \mathcal{F}_3$, and $\mathcal{F}_1 \& \mathcal{F}_3$ are larger than the terms in the sequence of $\mathcal{F}$. Hence we cannot omit any of these properties.

Approximation II: adding homogeneity. An SPG is homogenous (Theorem 1.2), i.e., 0-edges and 1-edges cannot enter nonterminal node at the same time. As the graph is Hamiltonian, we can say that if there exists a 0-edge $v_i \to v_j$, then the edge $v_{j-1} \to v_j$ is a 0-edge:

$$\mathcal{F}_4 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (x_{i,j} \to x_{j-1,j}).$$

Similarly, if there exists a 1-edge $v_i \to v_j$, then the edge $v_{j-1} \to v_j$ is a 1-edge:

$$\mathcal{F}_5 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (y_{i,j} \to y_{j-1,j}).$$

Now the approximation of the formula $\mathcal{F}_n^*$ for the class SPG is

$$\mathcal{F} \equiv \mathcal{F}_1 \& \mathcal{F}_2 \& \mathcal{F}_3 \& \mathcal{F}_4 \& \mathcal{F}_5$$

and the corresponding sequence for it is

$$1, 2, 6, 24, 120, 720, \ldots$$

Since the number of 4-node SPGs must be 22, it is obvious that in the set of satisfying assignments of the last formula there are 2 assignments that do not represent any SPG. Since we have not got any more properties, there must exist a yet undeclared property that reveals itself in case $n \ge 4$.

In order to understand the following properties better, we depict SPGs on figures, lining them up according to their Hamiltonian path.

Approximation III: specifying the homogeneity property. When analyzing the 24 assignments we received in case $n = 4$, we can see that 2 of them don not represent any SPG (Figure 4.1, the traditional form in Figure 4.2). In the first of

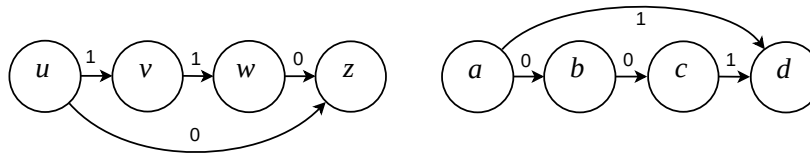


Figure 4.1: Forbidden situations in case $n = 4$.

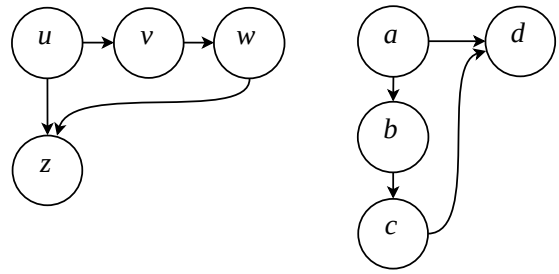


Figure 4.2: Forbidden situations in case $n = 4$ in traditional form.

them the 0-edge $v \to z$ is missing and in the second of them the 1-edge $b \to d$ is missing.

One possibility to describe the found property is to restrict formulae $\mathcal{F}_4$ and $\mathcal{F}_5$. Hence, we formulate the property that if there exists a 0-edge $v_i \to v_{j+1}$, then there also exists the 0-edge $v_j \to v_{j+1}$ and if for some $k = i+1, \ldots, j-1$ the path from node v_k to node v_j consists of only 1-edges, then there exists the 0-edge $v_k \to v_{j+1}$ (Figure 4.3):

$$\mathcal{F}_4 \equiv \bigwedge_{1 \le i < j \le n-1} (x_{i,j+1} \to x_{j,j+1} \& (\bigwedge_{i+1 \le k \le j-1} ((\bigwedge_{k \le p \le j-1} y_{p,p+1}) \to x_{k,j+1}))).$$

Dually: if there exists a 1-edge $v_i \to v_{j+1}$, then there exists the 1-edge $v_j \to v_{j+1}$

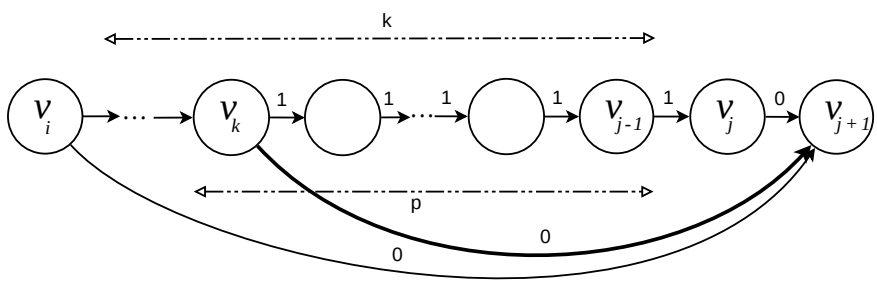


Figure 4.3: The specification of property $\mathcal{F}_4$ in Approximation III.

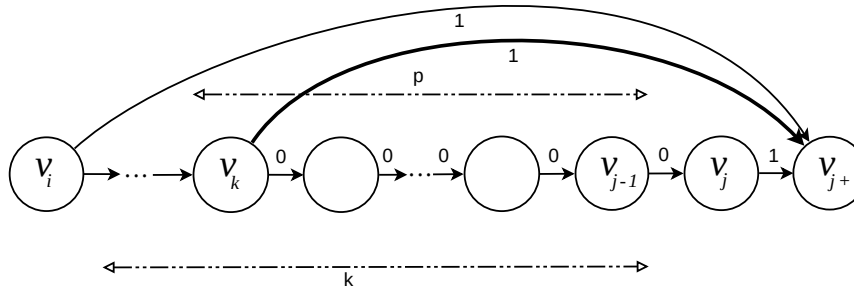


Figure 4.4: The specification of property $\mathcal{F}_5$ in Approximation III.

and if for some $k = i + 1, \ldots, j - 1$ the path from node v_k to node v_j consists of only 0-edges, then there exists the 1-edge $v_k \to v_{j+1}$ (Figure 4.4):

$$\mathcal{F}_5 \equiv \bigwedge_{1 \le i < j \le n-1} (y_{i,j+1} \to y_{j,j+1} \& (\bigwedge_{i+1 \le k \le j-1} ((\bigwedge_{k \le p \le j-1} x_{p,p+1}) \to y_{k,j+1}))).$$

Substituting in approximation $\mathcal{F}$ formulae $\mathcal{F}_4$ and $\mathcal{F}_5$ with their improved versions and running the translator and the counter, we get the sequence

$$1, 2, 6, 22, 94, 458, \ldots$$

We see that the situation is better (according to Sequence (2.1) the accurate sequence is $1, 2, 6, 22, 90, 394, \ldots$), but despite that, our description is still incorrect. One can guess that either the properties connected with the formulae $\mathcal{F}_4$ and $\mathcal{F}_5$ are more general than the one we described or some yet unknown property is missing.

Approximation IV: unfit assignments in case $n = 5$. In case $n = 5$ we find further 4 assignments that do not represent any SPG.

Let it be mentioned as a remark that since the properties expressed by the formulae $\mathcal{F}_4$ and $\mathcal{F}_5$ are at the moment unproved for SPGs, we are not absolutely certain that, from the 94 assignments found, precisely 90 are right and 4 wrong; there could be more wrong ones. To be sure, we have $94/2 = 47$ different graphs (others are duals of these, where 0- and 1-edges are exchanged); it is not hard to review them manually.

One of the problematic graphs (assignments) is presented in Figure 4.5. and *traditional view* corresponding to it is in Fig. 4.6. We can see that either the 0-edge $a \to d$ or the 1-edge $c \to e$ is in excess, or the 1-edge $b \to e$ is missing. Since it is easier to express the fact that an edge is missing (no need to use the negation), we choose to do this.

Therefore, we formulate the property that if a 0-edge and a 1-edge are crossing in a way shown in the Figure 4.7, then there must be an exiting 1-edge at every

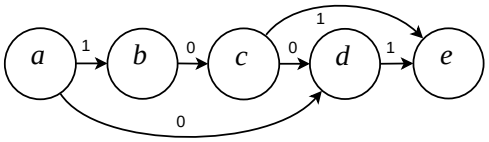


Figure 4.5: The forbidden situation found at Approximation III.

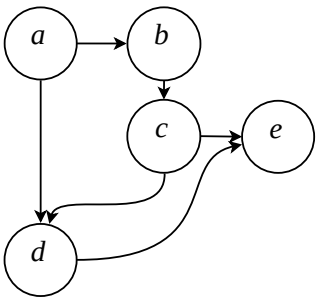


Figure 4.6: The forbidden situation found at Approximation III (in traditional form).

node between the startpoints of those edges, the endpoint of this exiting 1-edge lying no farther than v_r:

$$\mathcal{F}_6 \equiv \bigwedge_{1\le k<l<p<r\le n} ((x_{k,p}\&y_{l,r}) \to (\bigwedge_{k\le s\le(l-1)} (\bigvee_{(s+1)\le t\le r} y_{s,t}))).$$

Dual of this property: if a 0-edge and a 1-edge are crossing like shown in Figure 4.8, then there must be an exiting 0-edge at every node between the startpoints

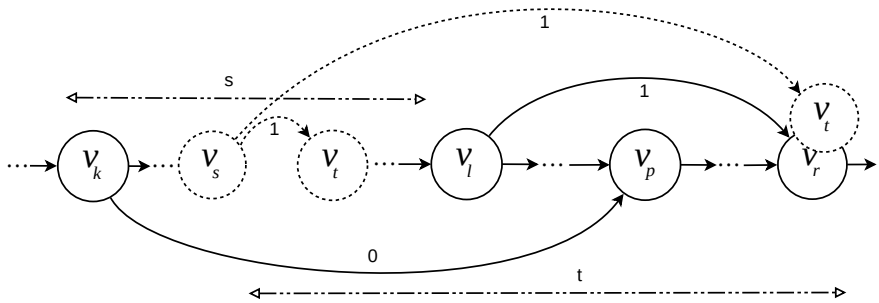


Figure 4.7: The property found at Approximation IV.

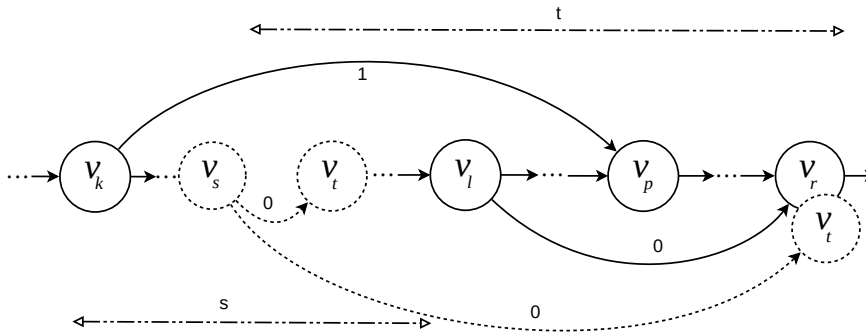


Figure 4.8: The property found at Approximation IV, dual case.

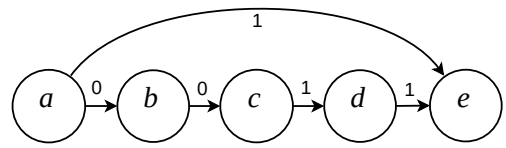


Figure 4.9: The situation forbidden by Approximation IV.

of those edges, with the endpoints of these exiting edges lying no farther than v_r:

$$\mathcal{F}_7 \equiv \bigwedge_{1\leq k<l<p<r\leq n} ((y_{k,p} \& x_{l,r}) \rightarrow (\bigwedge_{k\leq s\leq (l-1)} (\bigvee_{(s+1)\leq t\leq r} x_{s,t}))).$$

By adding the formulae $\mathcal{F}_6$ and $\mathcal{F}_7$ to the set of properties and running the translator and the counter, we get the integer sequence

$$1, 2, 6, 22, 92, 428, \ldots$$

Therefore, the formulae $\mathcal{F}_6$ and $\mathcal{F}_7$ do not completely describe the missing property and if $n = 5$, we again try to find the so far unfound 2 assignments α and β, where $\mathcal{F}(\alpha) = 1$ and $\mathcal{F}(\beta) = 1$ but the graphs defined by the assignments α ja β are not SPGs. Seeing the symmetry of formulae $\mathcal{F}_1, \ldots, \mathcal{F}_7$ with respect to 0- and 1-edges, it is clear that β is retrievable from α by exchanging the 0- and 1-edges.

Approximation V: specification of description on the basis of the case $n = 5$. By analyzing satisfying assignments acquired after adding the formulae $\mathcal{F}_6$ and $\mathcal{F}_7$ we find the assignment α that is presented as a graph in the Figure 4.9 with the traditional view in Figure 4.10. To generalize this, we formulate the property that for every 1-edge $v_i \rightarrow v_{j+1}$ and every node v_k that lies between the nodes v_i and v_{j+1}, if there exists a 0-edge $v_k \rightarrow v_{k+1}$, then there has to be a 1-edge going from the node v_k to the node that lies no farther than v_{j+1} (Figure 4.11):

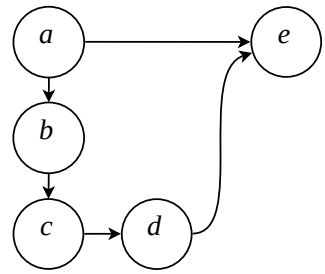


Figure 4.10: The situation forbidden by Approximation IV (in traditional form).

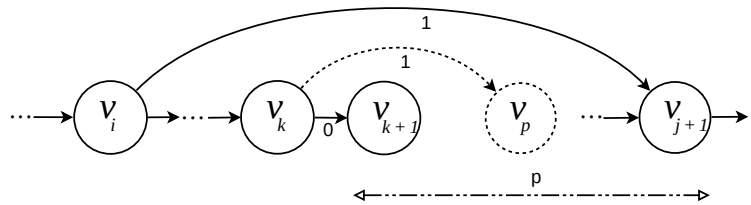


Figure 4.11: The found property by Approximation V.

$$\mathcal{F}_8 \equiv \bigwedge_{1\le i<j\le n-1} (y_{i,j+1} \to (\bigwedge_{i\le k\le j-1} (x_{k,k+1} \to (\bigvee_{k+1\le p\le j+1} y_{k,p})))).$$

Dual of the previous property: for every 0-edge $v_i \to v_{j+1}$ and every node v_k that lies between the startpoint and endpoint of this 0-edge, if there exists a 1-edge $v_k \to v_{k+1}$, then there has to be a 0-edge going from the node v_k to a node that lies no farther than node v_{j+1} (Figure 4.12):

$$\mathcal{F}_9 \equiv \bigwedge_{1\le i<j\le n-1} (x_{i,j+1} \to (\bigwedge_{i\le k\le j-1} (y_{k,k+1} \to (\bigvee_{k+1\le p\le j+1} x_{k,p})))).$$

When we add the properties $\mathcal{F}_8$ and $\mathcal{F}_9$ to the approximation $\mathcal{F}$ and use the translator and the counter, we get the sequence

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, \ldots$$

which coincides with the beginning of the sequence of large Schröder numbers.

Approximation VI: simplification of the sufficient set of properties. Using the translator and the counter, we can convince ourselves that the formulae $\mathcal{F}_8$ and $\mathcal{F}_9$ allow to simplify the formulae $\mathcal{F}_4$ and $\mathcal{F}_5$ to

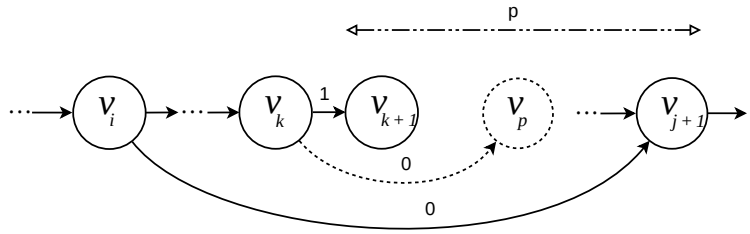


Figure 4.12: The found property by Approximation V, dual case.

$$\mathcal{F}_4 \equiv \bigwedge_{1\le i\le(n-2)} \bigwedge_{(i+2)\le j\le n} (x_{i,j} \rightarrow x_{j-1,j}),$$

$$\mathcal{F}_5 \equiv \bigwedge_{1\le i\le(n-2)} \bigwedge_{(i+2)\le j\le n} (y_{i,j} \rightarrow y_{j-1,j}).$$

As the last set of properties describes such family of superpositional formulae, which in case $n \le 9$ has exactly the same number of satisfying assignments as is the number of n-node SPGs, then probably the class SPG can be described by the formula

$$\mathcal{F} \equiv \mathcal{F}_1 \& \mathcal{F}_2 \& \mathcal{F}_3 \& \mathcal{F}_4 \& \mathcal{F}_5 \& \mathcal{F}_6 \& \mathcal{F}_7 \& \mathcal{F}_8 \& \mathcal{F}_9.$$

and without composition

$$\begin{aligned}
\mathcal{F} \equiv & \bigwedge_{1\le i\le n-1} xor(x_{i,i+1}, y_{i,i+1}) \\
\& & \bigwedge_{1\le i\le n-1} atmostone(x_{i,j} : i < j \le n) \\
\& & \bigwedge_{1\le i\le n-1} atmostone(y_{i,j} : i < j \le n) \\
\& & \bigwedge_{1\le i\le(n-2)} \bigwedge_{(i+2)\le j\le n} (x_{i,j} \rightarrow x_{j-1,j}) \\
\& & \bigwedge_{1\le i\le(n-2)} \bigwedge_{(i+2)\le j\le n} (y_{i,j} \rightarrow y_{j-1,j}) \\
\& & \bigwedge_{1\le k<l<p<r\le n} ((x_{k,p} \& y_{l,r}) \rightarrow (\bigwedge_{k\le s\le(l-1)} (\bigvee_{(s+1)\le t\le r} y_{s,t}))) \\
\& & \bigwedge_{1\le k<l<p<r\le n} ((y_{k,p} \& x_{l,r}) \rightarrow (\bigwedge_{k\le s\le(l-1)} (\bigvee_{(s+1)\le t\le r} x_{s,t})))
\end{aligned}$$

$$\&\bigwedge_{1\leq i<j\leq n-1}(y_{i,j+1}\rightarrow(\bigwedge_{i\leq k\leq j-1}(x_{k,k+1}\rightarrow(\bigvee_{k+1\leq p\leq j+1}y_{k,p}))))$$
$$\&\bigwedge_{1\leq i<j\leq n-1}(x_{i,j+1}\rightarrow(\bigwedge_{i\leq k\leq j-1}(y_{k,k+1}\rightarrow(\bigvee_{k+1\leq p\leq j+1}x_{k,p})))).$$

By using the translator and the counter we make sure that none of the properties can be omitted from the last approximation. It is clear that if the description is not correct, the inaccuracy or even a missing property can appear only in case $n \geq 10$.

The acquired approximation $\mathcal{F}$ is sophisticated and does not associate with classical properties of graphs. Hence, inevitably, the question arises whether our chosen set X of propositional variables is suitable. Even though the set X of propositional variables has been chosen in a traditional way, as has been done in previous articles describing SSBDDs (for reasons of clarity, edges pointing to terminal nodes were missing in the figures), we can see that the difficulty of expression and technicality were caused by our incapability to describe the properties of edges pointing directly to terminal nodes.

It will appear that the situation gets better if we add to the set X of propositional variables the variables, which describe the existence of edges pointing to the terminal nodes (Chapter 5).

CHAPTER 5

DESCRIPTION OF SUPERPOSITIONAL GRPAHS WITH TERMINAL NODES

In this chapter we find the description of the class SPG by means of propositional formulae taking into account the properties of edges pointing to terminal nodes.

5.1 Representation of Binary Graph by Propositional Variables

Similarly to the previous description (Section 4), there exists a unique Hamiltonian path in $G \in SPG$, which gives a canonical enumeration of the nodes of G. Let the Hamiltonian path of an n-node SPG G consist of nodes $v_1, \ldots, v_n$; and let v_{n+1} represent both the terminal 0 and the terminal 1. Such approach is not restrictive (but enables a simpler description of properties).

The meaning of the propositional variables $x_{i,j}$ and $y_{i,j}$ is, respectively, the existence of 0-edge and 1-edge from node v_i to node v_j:

$$x_{i,j} = \begin{cases} 1, & \text{if there exists a 0-edge } v_i \to v_j; \\ 0, & \text{otherwise;} \end{cases}$$

$$y_{i,j} = \begin{cases} 1, & \text{if there exists a 1-edge } v_i \to v_j; \\ 0, & \text{otherwise.} \end{cases}$$

Since SPGs are acyclic, then in case there exists an edge from node v_i to node v_j, certainly $i < j$. Therefore, propositional formulae for the class SPG are defined on the set

$$X = \{x_{i,j}, y_{i,j} : 1 \leq i < j \leq n+1\}$$

and the required formula $\mathcal{F}_n^*(X)$ should satisfy the condition

$$\mathcal{F}_n^*(\alpha) = \begin{cases} 1, & \text{if the graph represented by } \alpha \text{ is an SPG;} \\ 0, & \text{otherwise;} \end{cases}$$

for every assignment α to X.

5.2 Descriptive Properties of the Class SPG as Propositional Formulae

Approximation I: binary graphs with unique Hamiltonian path. The fact that the graph is acyclic is considered already when selecting the variables.

There is exactly one edge (0-edge or 1-edge) between the successive nodes of Hamiltonian path (except maybe between the last two nodes):

$$\mathcal{P}_1 \equiv \bigwedge_{1 \le i \le n-1} xor(x_{i,i+1}, y_{i,i+1}).$$

Each node (except the two last) has exactly one exiting 0-edge:

$$\mathcal{P}_2 \equiv \bigwedge_{1 \le i \le n-1} exactlyone(x_{i,j} : i < j \le n+1).$$

Each node (except the two last) has exactly one exiting 1-edge:

$$\mathcal{P}_3 \equiv \bigwedge_{1 \le i \le n-1} exactlyone(y_{i,j} : i < j \le n+1).$$

Taking into account that the previous properties must hold for every SPG, an approximation of $\mathcal{F}_n^*$ for the class SPG is

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3.$$

When applying the translator of propositional formulae to $\mathcal{F}$ and hereafter the counter of satisfying assignments to the result, we get the Sequence $A000165$:

$$1, 2, 8, 48, 384, 3840, 46080, \ldots$$

With the use of the translator and the counter, we can be convinced that the corrresponding members of sequences of formulae $\mathcal{P}_1 \& \mathcal{P}_2$, $\mathcal{P}_2 \& \mathcal{P}_3$ and $\mathcal{P}_1 \& \mathcal{P}_3$ are larger than in case of $\mathcal{F}$. Hence, we cannot skip any of these properties.

Approximation II: adding homogeneity. Every SPG is homogenous (Theorem 1.2), i.e., 0-edges and 1-edges cannot enter the same internal node at the same

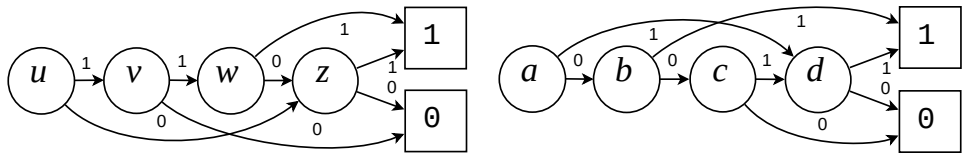


Figure 5.1: Forbidden situations in case $\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_4 \& \mathcal{P}_5$.

time. As there exists also the Hamiltonian path, we can equally say that if a 0-edge $v_i \to v_j$ exists, then the edge $v_{j-1} \to v_j$ is a 0-edge:

$$\mathcal{P}_4 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (x_{i,j} \to x_{j-1,j}).$$

Similarly, if there exists a 1-edge $v_i \to v_j$, then the edge $v_{j-1} \to v_j$ is a 1-edge:

$$\mathcal{P}_5 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (y_{i,j} \to y_{j-1,j}).$$

Now the approximation of $\mathcal{F}_n^*$ for the class SPG is

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_4 \& \mathcal{P}_5$$

and the corresponding sequence for it is $A000142$:

$$1, 2, 6, 24, 120, 720, \ldots$$

However, while the number of 4-node SPGs must be 22, it is obvious that in the set of received assignments there are 2 assignments that do not represent any SPG. Therefore, there must exist a yet undeclared property that *appears* in case $n \ge 4$.

Approximation III: adding the strong planarity. Analyzing the 24 assignments we received in case $n = 4$, we see that 2 of them do not represent any SPG (Figure 5.1). We guess that, in either case, the problem is caused by crossing same type of edges. We raise a hypothesis that edges of the same type can not cross and add the corresponding properties to the approximation.

Let $1 \le k < l < p < r \le n + 1$. According to the hypothesis, there can not be a situation where there the 0-edges $v_k \to v_p$ and $v_l \to v_r$ exist at the same time, i.e., 0-edges do not cross (Figure 5.2):

$$\mathcal{P}_6 \equiv \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{x_{k,p} \& x_{l,r}}).$$

Herewith dually, 1-edges do not cross (Figure 5.2):

$$\mathcal{P}_7 \equiv \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{y_{k,p} \& y_{l,r}}).$$

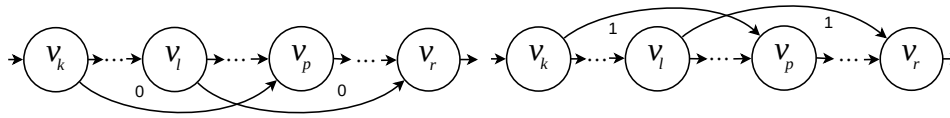


Figure 5.2: Situations forbidden by the properties $\mathcal{P}_6$ and $\mathcal{P}_7$.

Definition 5.1. We say that edges $v_k \to v_p$ and $v_l \to v_r$ are *crossing* edges if $k < l < p < r$.

Definition 5.2. We say that a binary traceable graph is *strongly planar* if it has no crossing 0-edges and no crossing 1-edges.

Obviously, the properties $\mathcal{P}_6$ and $\mathcal{P}_7$ express the strong planarity. In Figure 6.2 [9] we see that if a binary graph is strongly planar, it is also planar. The opposite is generally not true. Now for the formula

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_4 \& \mathcal{P}_5 \& \mathcal{P}_6 \& \mathcal{P}_7$$

we get the Sequence $A001181$:

$$1, 2, 6, 22, 92, 422, \ldots$$

We see that the situation is better, but our description is nevertheless incorrect (the accurate sequence is $1, 2, 6, 22, 90, 394, \ldots$). One could suppose that either some problem connected with the described properties is more general than the one we described or some yet unknown property is missing.

Approximation IV: removing the homogeneity. By using the translator and the counter, we can see that one can omit the properties $\mathcal{P}_4$ and $\mathcal{P}_5$ from the set of properties. Consequently, we can propose a hypothesis that the formula

$$\mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7 \longrightarrow \mathcal{P}_4 \& \mathcal{P}_5$$

is a tautology. We formulate it as the Theorem 5.1.

Theorem 5.1. If a binary traceable graph is strongly planar, then it is also homogenous.

We will prove it later (Theorem 6.5). Hence, the new approximation for the formula $\mathcal{F}_n^*$ is

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7.$$

Approximation V: adding the cofinality. We analyze the two assignments emerging from the last approximation in case $n = 5$. One of the found problematic graphs is depicted in Figure 5.3. The second problematic graph is obviously dual to the first one, i.e., it can be found from the other by exchanging 0- and 1-edges.

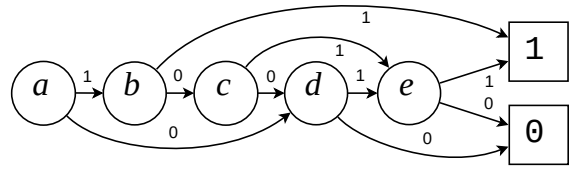


Figure 5.3: A forbidden situation in case $\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7$.

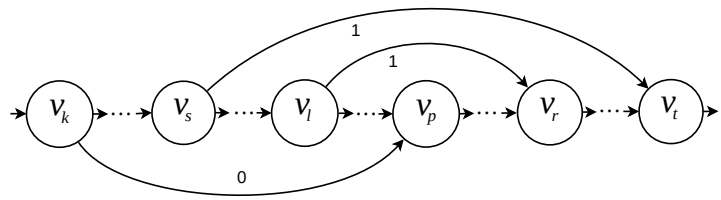


Figure 5.4: Situation forbidden by the property $\mathcal{P}_8$.

We suppose that, due to existence of the 0-edge $a \to d$, the 1-edges $b \to 1$ and $c \to e$ should point to the same node, i.e., the situation in Figure 5.4 is forbidden:

$$\mathcal{P}_8 \equiv \bigwedge_{1 \le k<s<l<p<r<t \le n+1} (\overline{x_{k,p} \& y_{l,r} \& y_{s,t}}).$$

Analogically: all 0-edges, which cross the same 1-edge, must point to the same node:

$$\mathcal{P}_9 \equiv \bigwedge_{1 \le k<s<l<p<r<t \le n+1} (\overline{y_{k,p} \& x_{l,r} \& x_{s,t}}).$$

Definition 5.3. We say that a binary traceable graph is 1-*cofinal (*0-*cofinal)* if all 1-edges (0-edges), starting between the endpoints of some 0-edge (1-edge) and crossing it, are entering into the same node.

Definition 5.4. We say that a binary traceable graph is *cofinal* if it is 1-cofinal and 0-cofinal.

After adding the properties $\mathcal{P}_8$ and $\mathcal{P}_9$ expressing cofinality to the approximation and running the counter, we get the integer sequence

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, \ldots$$

which also coincides with the beginning of the sequence of large Schröder numbers. Using the translator and the counter, we establish that none of the properties in the set $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_6, \mathcal{P}_7, \mathcal{P}_8, \mathcal{P}_9\}$ can be omitted.

We checked that in case $n = 1 \ldots 9$, the constructed set of properties has exactly the same number of satisfying assignments as is the number of n-noded SPGs. Hence, the class SPG can probably be described by the formula

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7 \& \mathcal{P}_8 \& \mathcal{P}_9$$

or written directly, not using composition:

$$\begin{aligned}
\mathcal{F} \equiv & \bigwedge_{1\le i\le n-1} xor(x_{i,i+1}, y_{i,i+1}) \\
& \& \bigwedge_{1\le i\le n-1} exactlyone(x_{i,j} : i < j \le n+1) \\
& \& \bigwedge_{1\le i\le n-1} exactlyone(y_{i,j} : i < j \le n+1) \\
& \& \bigwedge_{1\le k<l<p<r\le n+1} (\overline{x_{k,p}\& x_{l,r}}) \\
& \& \bigwedge_{1\le k<l<p<r\le n+1} (\overline{y_{k,p}\& y_{l,r}}) \\
& \& \bigwedge_{1\le k<s<l<p<r<t\le n+1} (\overline{x_{k,p}\& y_{l,r}\& y_{s,t}}) \\
& \& \bigwedge_{1\le k<s<l<p<r<t\le n+1} (\overline{y_{k,p}\& x_{l,r}\& x_{s,t}}).
\end{aligned}$$

It is clear that if the description is not correct, then an inaccuracy or even a missing property can appear only in case $n \ge 10$. To be convinced in the correctness of the formula, we formulate the following hypothesis.

Hypothesis. A binary graph is a superpositional graph iff it is a strongly planar cofinal traceable graph.

We prove this hypothesis in the next chapter.

5.3 Open Problems

All the approximations are characterized by sequences that match those from the Encyclopedia of Integer Sequences [10]. This indicates that these subclasses of binary graphs may have some connections with mathematical structures described by found sequences. It would be interesting, for example, to find bijections between

- double-downgrading permutations and traceable binary graphs ($A000165$);
- permutations and homogeneous binary graphs ($A000142$);
- Baxter permutations and strongly planar binary graphs ($A001181$);
- separable permutations and superpositional graphs ($A006318$).

CHAPTER 6

THEORY OF SUPERPOSITIONAL GRAPHS

This chapter contains definitions and theorems, which are necessary to prove the main theorem at the end of this section. We also reproduce some material from previous chapters for the clearness of the presentation. The results presented in this chapter are published in [9].

Definition 6.1. The *class* SPG of graphs is defined inductively as follows:

1° binary graphs $A, C, D \in SPG$;

2° if $G, H \in SPG$ and v is an internal node of G, then $G_{v \leftarrow H} \in SPG$.

Definition 6.2. A 0-*path (*1-*path)* from node u to node v is a path, which contains only 0-edges (1-edges).

Theorems 6.1, 6.2, and 6.3 were proved in the Section 1.2 (see Theorem 1.2).

Theorem 6.1. Let $G \in SPG$. Then for every internal node u there exist a 0-path $u \rightsquigarrow 0$ and a 1-path $u \rightsquigarrow 1$.

Definition 6.3. We say that a binary graph G is *homogenous* if only one type of edges enter into every node $v \in V(G)$.

Theorem 6.2. Every superpositional graph is homogenous.

Definition 6.4. A binary graph G is *traceable* if there exists a path through all internal nodes of G (Hamiltonian path).

Binary graphs are acyclic, therefore, if the Hamiltonian path exists, then it is unique.

Theorem 6.3. Every superpositional graph is traceable.

Theorem 6.3 gives a canonical enumeration of the nodes of a superpositional graph.

Definition 6.5. We say that a binary graph G has the *triangle property* if for every three internal nodes x,y, and z the existence of a 1-path $x \rightsquigarrow y$ and a 0-path $x \rightsquigarrow z$ implies the existence of either a 1-path $z \rightsquigarrow y$ or a 0-path $y \rightsquigarrow z$.

Theorem 6.4. Every superpositional graph has the triangle property.

Proof. We prove the theorem by induction over the structure of superpositional graph.

Basis of induction: graphs A, C, and D have the triangle property.

Induction step: Let $S = G_{w \leftarrow H}$. By induction hypothesis the triangle property holds for the graphs G and H.

Let x, y, and z be internal nodes of the graph S. This means that each of them is an internal node of either graph G or graph H and none of them is w. There are 8 possibilities choose the nodes x, y and z.

1. $x, y, z \in H$. By induction hypothesis, the assertion holds.

2. $x, y, z \in G$. We prove that if there exists a 0-path or a 1-path between the nodes y and z in the graph G, then this path also exists in the graph $G_{w \leftarrow H}$. According to the Theorem 6.1 there exist a 0-path $root(H) \rightsquigarrow low(w)$ and a 1-path $root(H) \rightsquigarrow high(w)$ in the graph $G_{w \leftarrow H}$. It is obvious that if some 0-path (1-path) goes through the node w, then this path also goes through the node $low(w)$ ($high(w)$). Therefore, if some 0-path $y \rightsquigarrow z$ goes through the node w (then also to the node $low(w)$), then a 0-path $y \rightsquigarrow z$ exists also in the graph $G_{w \leftarrow H}$. Similarly, if there exists a 1-path $z \rightsquigarrow y$ between the nodes y and z in the graph G, then there is a 1-path $z \rightsquigarrow y$ also in the graph $G_{w \leftarrow H}$.

3. $x, y \in H, z \in G$. Since there exists a 0-path $x \rightsquigarrow z$ in the graph $G_{w \leftarrow H}$, there also exists a 0-path $x \rightsquigarrow low(w)$. According to the Theorem 6.1 there exists a 0-path to the node 0 from each internal node of H, therefore there exists a 0-path $y \rightsquigarrow low(w)$ in the graph S. Hence there exists a 0-path $y \rightsquigarrow z$ in S.

4. $x \in H, y \in G, z \in H$. Dual to the case 3.

5. $x \in H, y, z \in G$. According to the definition of superpositional graphs, the 1-path $x \rightsquigarrow y$ goes through the node $high(w)$ and the 0-path $x \rightsquigarrow z$ goes through the node $low(w)$. Therefore there exists a 1-path $w \rightsquigarrow y$ and a 0-path $w \rightsquigarrow z$ in graph G. By induction hypothesis, there exists a 1-path or a 0-path between y and z in $G \backslash \{w\}$ (because G is acyclic) and hence in S.

6. $x \in G, y, z \in H$. The 1-path $x \rightsquigarrow y$ and 0-path $x \rightsquigarrow z$ must go through the unique root of H, which is impossible because superpositional graphs are homogenous.

7. $x \in G, y \in H, z \in G$. Since there exists a 1-path $x \rightsquigarrow y$ in S, there exists a 1-path $x \rightsquigarrow w$ in G. Using the triangle property for G, there is either a 0-path $w \rightsquigarrow z$ or a 1-path $z \rightsquigarrow w$ in G. If there exists a 0-path $w \rightsquigarrow z$ in the graph G and since there exists a 0-path $y \rightsquigarrow low(w)$ (by Theorem 6.1) and a 0-path $low(w) \rightsquigarrow z$, there also exists a 0-path $y \rightsquigarrow z$ in the graph $G_{w \leftarrow H}$. If there exists a 1-path $z \rightsquigarrow w$ in the graph G, then there exists a 1-path $z \rightsquigarrow root(H)$ and a 1-path $root(H) \rightsquigarrow y$, which is the 1-path $z \rightsquigarrow y$ in the graph $G_{w \leftarrow H}$.

8. $x, y \in G, z \in H$. Dual to case 7. □

It is easy to see that every superpositional graph is planar. We can prove an even more restrictive property of superpositional graphs.

Definition 6.6. The edges $v_k \rightarrow v_p$ and $v_l \rightarrow v_r$ of a binary traceable graph are *crossing edges* if $k < l < p < r$.

Definition 6.7. We say that a binary traceable graph is *strongly planar* if there are no crossing 0-edges and no crossing 1-edges.

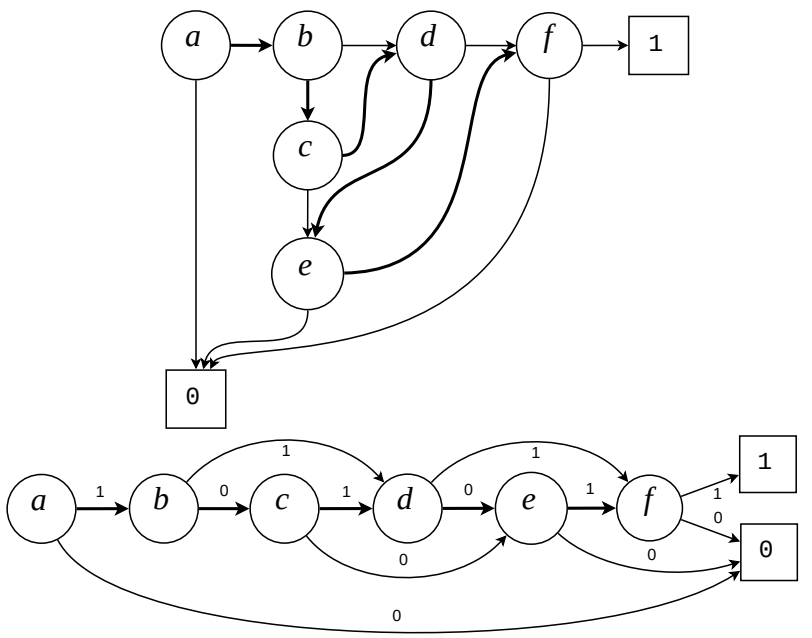


Figure 6.1: A superpositional graph before and after stretching. Bold arrows mark the Hamiltonian path.

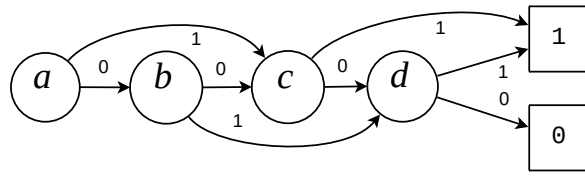


Figure 6.2: A planar graph, which is not strongly planar.

The strong planarity has a nice graph-theoretical interpretation: stretch the graph so that all nodes are in straight line in canonical order and there are no 0-edges above the line and no 1-edges below the line. If the binary graph is strongly planar, then there are no crossing edges with the same label in such drawing. Figure 6.1 depicts a superpositional graph before and after stretching.

It is also obvious that if a binary graph is strongly planar, then it is also planar, while the opposite does not hold in general. In Figure 6.2 there is a binary graph, which is planar, but not strongly planar (as 1-edges are crossing).

Theorem 6.5. A binary traceable graph G is strongly planar iff G is homogenous and has the triangle property.

Proof. We prove this theorem in three parts.

1. *If G is strongly planar, then the triangle property holds in the graph G.* Let G be strongly planar and v_i, v_j and v_k be three distinct internal nodes such that there exists a 1-path $v_i \rightsquigarrow v_j$ and a 0-path $v_i \rightsquigarrow v_k$. Let $k < j$. Start moving forward from the node v_k using only 1-edges. Eventually we will reach the 1-path $v_i \rightsquigarrow v_j$, since the 1-edges do not cross. Thus there exists a 1-path $v_k \rightsquigarrow v_j$. If $k > j$, then there analogously exists a 0-path $v_j \rightsquigarrow v_k$.

2. *If G is strongly planar, then G is homogenous.* Let G be strongly planar. We assume by contradiction that both a 0-edge and a 1-edge enter into some internal node v_j. It is obvious that one of these edges is $v_{j-1} \to v_j$. Let it be 0-edge (in case of 1-edge the proof is similar). Now we show that in this case there exist two crossing 1-edges. Since the graph is binary, there exists a 1-edge $v_{j-1} \to v_m$, where $m > j$ but in this case the graph G is not strongly planar as this edge crosses the 1-edge coming into v_j (see Figure 6.3).

3. *If G is homogenous and has the triangle property, then G is strongly planar.* Let G be a traceable homogenous binary graph with the triangle property. We assume by contradiction that it is not strongly planar. Suppose there are crossing 1-edges in G. Let $i < j < k < m$ be the indexes such that 1-edges $v_i \to v_k$ and $v_j \to v_m$ are crossing, where j is minimal and i is maximal i.e. the choice of i depends on j (Figure 6.4). With this we guarantee, that any 1-edge beginning between nodes v_i and v_j does not cross the 1-edge $v_j \to v_m$.

Due to the triangle property and acyclicity there is a 1-path $v_{i+1} \rightsquigarrow v_k$. This 1-path must contain v_j because of our choice of i, j, k, and m. But then two 1-edges start from v_j, which contradicts the definition of a binary graph. □

Corollary 6.1. Every superpositional graph is strongly planar.

Proof. Immediate consequence ot Theorems 6.2, 6.3, 6.4, and 6.5. □

Definition 6.8. We say that a binary traceable graph is 1-*cofinal (*0-*cofinal)* if all 1-edges (0-edges), starting between the endpoints of some 0-edge (1-edge) and crossing it, end in the same node.

Figures 6.5 and 6.6 illustrate these notions.

Definition 6.9. A binary traceable graph is *cofinal* if it is 1-cofinal and 0-cofinal.

Theorem 6.6. Every superpositional graph is cofinal.

Proof. We prove the 1-cofinality, the proof of the 0-cofinality is similar. For the verification of 1-cofinality it is enough to show that if there is a pair of 1-edges, starting between the endpoints of some 0-edge and crossing it, then these 1-edges must have a common endpoint. Let us suppose that we have a 1-cofinal superpositional graph G and a 0-edge $a \to u$ in it. The only way to obtain, applying superposition, two 1-edges that start between a and u and cross $a \to u$ is to superpose a graph H instead of some b between a and u, which is a starting point of some 1-edge, crossing $a \to u$ (Figure 6.7). All internal edges of H remain unchanged. Only new edges which will cross the 0-edge are the 1-edges, which were pointing to the terminal 1 of H, but in $G_{b \leftarrow H}$ were redirected to $w = high(b)$. If graphs G and H are 1-cofinal, then so is $G_{b \leftarrow H}$. It means that the 1-cofinality is closed under superposition. □

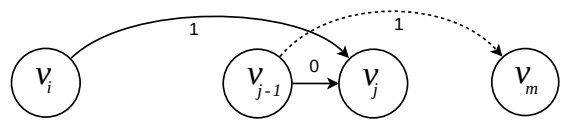


Figure 6.3: An illustrative figure for the proof of the Theorem 6.5 part 2.

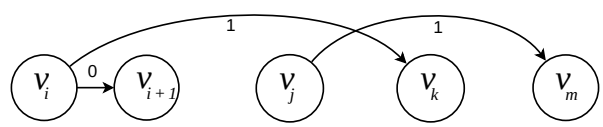


Figure 6.4: The edges $v_i \to v_k$ and $v_j \to v_m$ are crossing.

Lemma 6.1. If G is a traceable strongly planar cofinal binary graph with $n > 2$ internal nodes, then it can be represented as a superposition $G = H_{w \leftarrow F}$, where H and F are binary graphs with at least 2 nodes.

Proof. Let $v_1, \ldots, v_n$ be the canonical sequence of internal nodes of the graph G. We are looking for a subsequence $v_k, v_{k+1}, \ldots, v_l$ $(k < l)$ such that:

1. all edges from nodes $v_1, \ldots, v_{k-1}$ to the nodes of the subsequence are pointing to v_k;
2. all 1-edges from the nodes of the subsequence to nodes $v_{l+1}, \ldots, v_n, 1$ are pointing to the same node;
3. all 0-edges from the nodes of the subsequence to nodes $v_{l+1}, \ldots, v_n, 0$ are pointing to the same node.

We construct the binary graphs H and F, using the subsequence. The set of nodes of the graph H is $V(H) = \{v_1, \ldots, v_{k-1}, w, v_{l+1}, \ldots, v_n, 0, 1\}$. Edges of graph H are all edges of G with both endpoints in $V(H)$; all edges, pointing to v_k are redirected to the node w and the 1-edges (0-edges) from some node of subsequence to a node from the set $\{v_{l+1}, \ldots, v_n, 0, 1\}$ replaced by the 1-edge (0-edge) in H from w to the same node. Formally:

$$\begin{aligned} E(H) \;=\; & \{(u,v) :\; u,v \in V(H) \setminus \{w\}, (u,v) \in E(G)\} \cup \\ & \{(u,w) :\; u \in \{v_1, \ldots, v_{k-1}\}, (u,v_k) \in E(G)\} \cup \\ & \{(w,z) :\; z \in \{v_{l+1}, \ldots, v_n, 0, 1\}, \exists i_{k \le i \le l}((v_i, z) \in E(G))\}. \end{aligned}$$

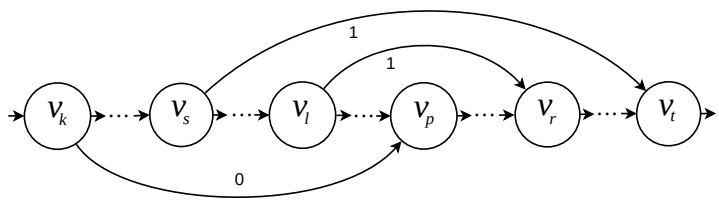


Figure 6.5: Situation forbidden by 1-cofinality.

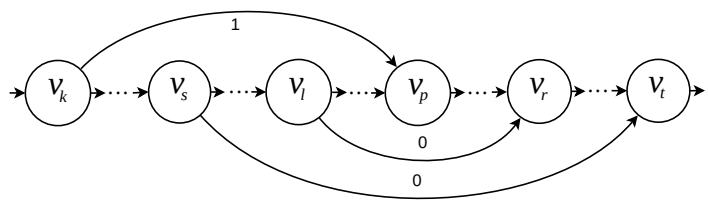


Figure 6.6: Situation forbidden by 0-cofinality.

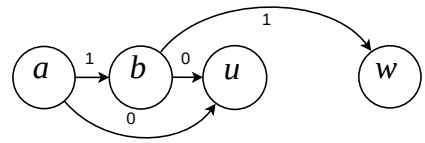


Figure 6.7: A fragment of a graph with crossing 0-edge and 1-edge.

The set of nodes of the graph F is $V(F) = \{v_k, \ldots, v_l, 0, 1\}$. The edges of the graph F are all edges of G with endpoints in $V(F)$; all 1-edges of G, going from $V(F)$ to the nodes from $\{v_{l+1}, \ldots, v_n, 1\}$ will be redirected to the terminal 1 and all 0-edges of G, going from $V(F)$ to the nodes from $\{v_{l+1}, \ldots, v_n, 0\}$, will be redirected to terminal 0 of F. Formally:

$$\begin{aligned} E(F) \quad = \quad & \{(u,v) : \ u,v \in V(F), (u,v) \in E(G)\} \cup \\ & \{(v_i,1) : \ v_i \in V(F), \exists z_{z \in \{v_{l+1},\ldots,v_n,1\}}(high(v_i) = z)\} \cup \\ & \{(v_i,0) : \ v_i \in V(F), \exists z_{z \in \{v_{l+1},\ldots,v_n,0\}}(low(v_i) = z)\}. \end{aligned}$$

Obviously, the construction of graphs H and F in conditions 1–3 is the reverse engineering of the definition of superposition of binary graphs. Therefore, $G = H_{w \leftarrow F}$. There are four cases the following.

1. $high(v_1) = 1$. Then $low(v_1) = v_2$ and the subsequence is $v_2, \ldots, v_n$.

2. $low(v_1) = 0$. Then $high(v_1) = v_2$ and the subsequence is $v_2, \ldots, v_n$.

3. $high(v_1) = v_{l+1}$, where $1 < l \leq n-1$. Then $low(v_1) = v_2$ and the subsequence is $v_1, \ldots, v_l$. Condition 1 is fulfilled trivially. Condition 2 is fulfilled, because if there is some 1-edge (v_i, z), where $2 \leq i \leq l, z \in \{v_{l+2}, \ldots, v_n, 1\}$, then we have two crossing 1-edges and G is not strongly planar. Condition 3 is fulfilled, because if there are two 0-edges, (v_i, u) and (v_j, z), where $2 \leq i < j \leq l$ and $u, z \in \{v_{l+2}, \ldots, v_n, 0\}$, $u \neq z$, then G is not cofinal.

4. $low(v_1) = v_{l+1}$, where $1 < l \leq n-1$. Then $high(v_1) = v_2$ and the subsequence is $v_1, \ldots, v_l$. Conditions 1–3 are fulfilled by considerations, symmetrical to the case 3. □

Theorem 6.7. A binary graph is superpositional graph iff it is a traceable strongly planar cofinal graph.

Proof. $\Longrightarrow$ Proved by Theorem 6.3, Corollary 6.1, and Theorem 6.6.

$\Longleftarrow$ We prove the proposition by induction over the number of nodes of the graph.

Basis of induction. There are two 2-node binary graphs. These are binary graphs C and D (Figure 1.1), which are superpositional graphs.

Induction step. Let G be a n-node traceable strongly planar cofinal graph ($n > 2$). By Lemma 6.1 it can be represented as a superposition of two graphs: $G = H_{w \leftarrow F}$, where $|V(H)| < |V(G)|$ and $|V(F)| < |V(G)|$. Graphs H and F inherite the properties of traceability, strong planarity and cofinality from G. By induction hypothesis H and F are superpositional graphs. Graph G is a superposition of two superpositional graphs and by Definiton 6.1 it is a superpositional graph. □

CHAPTER 7

SOLUTIONS TO THE PROBLEMS RAISED

In this chapter we give solutions to the problems raised in Chapter 2.

Problem I: testing whether a binary graph G is an SPG.

Using the Algorithm 1 (Section 1.2) we find the Hamiltonian path of the graph G. If it does not exist, then $G \notin SPG$. Otherwise we check, whether G is strongly planar and cofinal. If yes, then $G \in SPG$, otherwise $G \notin SPG$ (Theorem 6.7).

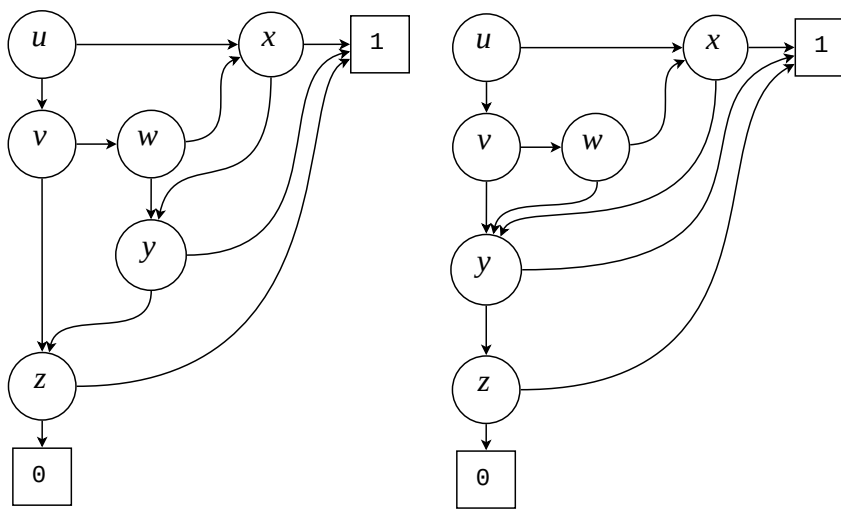


Figure 7.1: Two binary graphs.

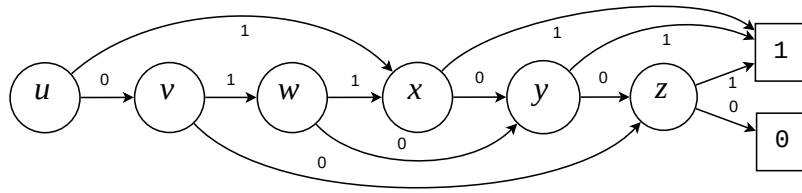


Figure 7.2: Leftmost graph from Figure 7.1 in canonical order.

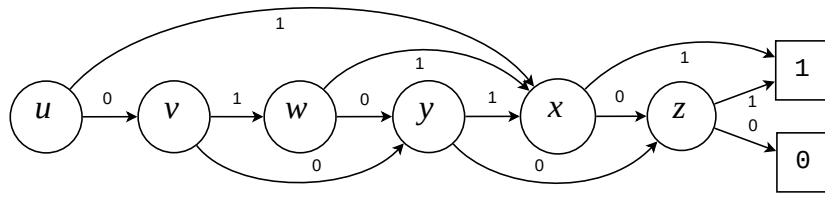


Figure 7.3: Rightmost graph from Figure 7.1 in canonical order.

For example, in Figure 7.1 there are two binary graphs. To check whether they are superpositional graphs, we stretch them by the canonical ordering of nodes. The leftmost graph on the Figure 7.1, is depicted in Figure 7.2 and the rightmost graph in Figure 7.3. It is obvious that both are traceable and strongly planar. The graph in the Figure 7.2 is not cofinal: nodes v and w lie between the endpoints of the 1-edge $u \to x$ but 0-edges $v \to z$ and $w \to y$ point to different nodes. According to the Theorem 6.7, the graph is not a superpositional graph. The graph in Figure 7.3 is cofinal and, therefore, it is a superpositional graph.

Algorithm 2 below shows that the test for crossing 1-edges can be done within time $\mathcal{O}(n)$. Since the test for crossing 0-edges is dual, we get the estimation for testing the strong planarity. Let Q be a stack, $\sigma_1(i)$ be the number of the node $high(v_i)$ in the canonical enumeration of the nodes (terminal node 1 has the number $n+1$). Let $v_i.d_1$ be the number of the long 1-edges pointing to v_i (the 1-edges $k \to i$, where $k+1 < i$).

Algorithm 2. For testing the existence of crossing 1-edges.

find_1_edges_cross (binary graph G){
  $Q = \{\}$;
  for $i = 1 \ldots n$**{**
    while ($v_i.d_1 > 0$) {
      p=Q.**pop**();
      if ($p > i$) **return** "Crossing";
      if ($p == i$) $v_i.d_1$--;
    }

if$(\sigma_1(i) > i+1)$ Q.**push**$((\sigma_1(i))_1)$;
//if from the node v_i exits a long 1-edge
//then push the number of it's endpoint into the stack
} *//for*
return "Do not cross";
}

The algorithm for testing the 1-cofinality is similar to the previous one. Testing the 0-cofinality is dual. Let Q be a stack, $\sigma_1(i)$ be the number of the node $high(v_i)$ and $\sigma_0(i)$ be the number of the node $low(v_i)$ in the canonical enumeration of the nodes (terminal node 1 has the number $n+1$). Let $v_i.d_0$ be the number of the long 0-edges pointing to v_i (the 0-edges $k \to i$, where $k+1 < i$).

Algorithm 3. For testing the 1-cofinality.

find_1_cofinal (binary strongly planar graph G){
$Q = \{\}$;
for $i = 1 \ldots n$**{**
while $(v_i.d_0 > 0)$ {
$target\_node = -1$;
//the endpoint of 1-edges, starting between
//the endpoints of the 0-edge and crossing it
p_x=Q.**pop**();
if$((x == 1)$**and**$(p > i))${ *//the endpoint of the 1-edge*
if $(target\_node == -1)$ $target\_node = p$;
//the first such edge
if $(target\_node <> p)$ **return** "Not cofinal";
//two 1-edges, which are not pointing to the same node
}
if$((x == 0)$**and**$(p == i))$ $v_i.d_0$--;
}
//if from the node v_i exits a long edge
//then push the number of it's endpoint into the stack
if$(\sigma_1(i) > i+1)$ Q.**push**$((\sigma_1(i))_1)$;
if$(\sigma_0(i) > i+1)$ Q.**push**$((\sigma_0(i))_0)$;
}
return "Cofinal";
}

To estimate the time complexity of a problem "whether a binary graph G is an SPG", let $|V(G)| = n$. Then time complexities of the operations are as follows.

- checking for binarity: $\mathcal{O}(n)$;
- finding the unique Hamiltonian path (Section 1.2): $\mathcal{O}(n)$;
- checking for strong planarity (Algorithm 2): $\mathcal{O}(n)$;
- checking for cofinality (Algorithm 3): $\mathcal{O}(n)$.

Hence the complexity of the algorithm is $\mathcal{O}(n)$.

Problem II: testing whether $G_1, G_2 \in SPG$ are isomorphic.

We can suppose that $|V(G_1)| = |V(G_2)|$, otherwise the graphs G_1 and G_2 are obviously not isomorphic.

By means of Algorithm 1 we find the Hamiltonian paths in the graphs G_1 and G_2, let these be $v_1, \ldots, v_n$ and $u_1, \ldots, u_n$, respectively. Let π be the function, which maps the node of the graph to its index in the Hamiltonian path. Now we check whether for every i ($1 \leq i \leq n$):

$$\pi(low(v_i)) = \pi(low(u_i)),$$

$$\pi(high(v_i)) = \pi(high(u_i)).$$

If yes, then G_1 and G_2 are isomorphic, otherwise they are not.

Hence the time complexity of the isomorphism is

$$\mathcal{O}(n) + 2 \cdot \mathcal{O}(n) = \mathcal{O}(n).$$

Problem III: finding a sequence of superpositions that generates a graph $G \in SPG$.

Let $G \in SPG$ and $v_1, \ldots, v_n$ be the canonical sequence of nodes of the graph G. According to the proof of Lemma 6.1, there exists a subsequence v_k, v_{k+1}, …, v_l ($k < l$) such that

- all edges from nodes v_1 ,…, v_{k-1} to the nodes of the subsequence are pointing to v_k;
- all 1-edges from the nodes of the subsequence to nodes v_{l+1}, …, v_n, 1 are pointing to the same node;
- all 0-edges from the nodes of the subsequence to nodes v_{l+1}, …, v_n, 0 are pointing to the same node.

This gives us a potential place of decomposition of the graph G. By applying the same procedure recursively to the parts, we eventually reach the elementary graphs C and D (Figure 1.1).

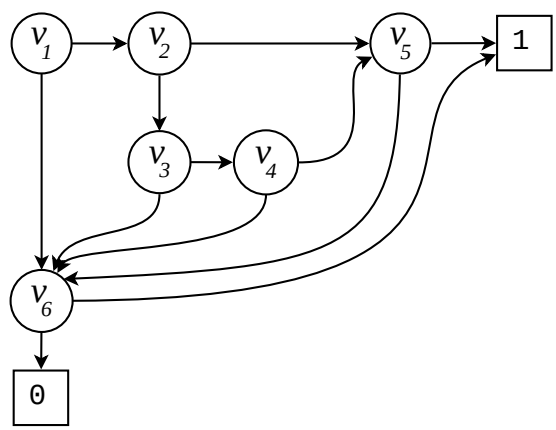
The following algorithm gives out a potential sequence of superpositions to generate the graph $G \in SPG$.

Algorithm 4. Finding the sequence of superpositions that generates $G \in SPG$.

decompose (superpositional graph G){
  if $G = C$ **or** $G = D$ **output** graph G;
  else {
    let $\{v_k, \ldots, v_l\}$ **and** w be as in the proof of Lemma 6.1;
    $V(F) := \{v_k, \ldots, v_l, 0, 1\}$;
    $V(H) := \{v_1, \ldots, v_{k-1}, w, v_{l+1}, \ldots, v_{|V(G)|}, 0, 1\}$;
    decompose(F);
    decompose(H);
    output $G = H_{w \leftarrow F}$;
  }
}

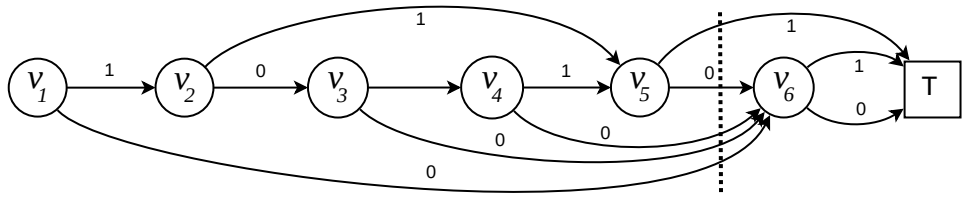
Obviously the number of superpositions for generating the graph $G \in SPG$ is not bigger than the number of nodes in G. By the proof of the Lemma 6.1, the place of decomposition can be found by constant time. Since we find less than n places of decomposition, the complexity of the algorithm is $\mathcal{O}(n)$.

Example. We find the sequence of superpositions that generates the following graph:



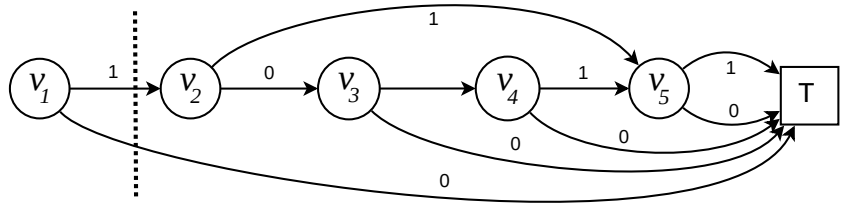
Step I: decompose(G)

$G = (G_1)_{v_5 \leftarrow G_2}$, where $V(G_2) = \{v_1, \ldots, v_5\}$; $V(G_1) = \{v_5, v_6\}$



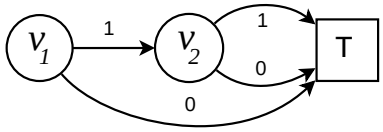
Step II: decompose(G_2)

$G_2 = (G_3)_{v_2 \leftarrow G_4}$, where $V(G_3) = \{v_1, v_2\}$; $V(G_4) = \{v_2, \ldots, v_5\}$



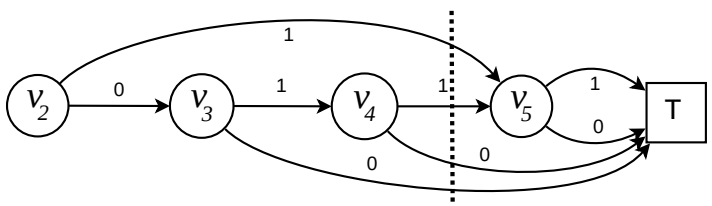
Step III: decompose(G_3)

$G_3 = C$ (Fig. 1.1), where $V(C) = \{v_1, v_2\}$



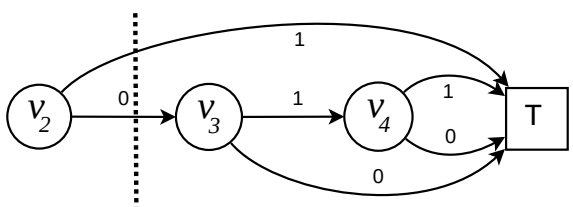
Step IV: decompose(G_4)

$G_4 = (G_5)_{v_4 \leftarrow G_6}$, where $V(G_6) = \{v_2, \ldots, v_4\}$; $V(G_5) = \{v_4, v_5\}$

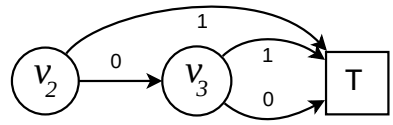


Step V: decompose(G_6)

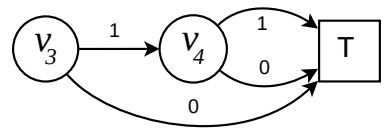
$G_6 = (G_7)_{v_3 \leftarrow G_8}$, where $V(G_7) = \{v_2, v_3\}$; $V(G_8) = \{v_3, v_4\}$



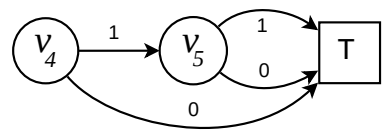
Step VI: decompose(G_7)
$G_7 = D$ (Fig. 1.1), where $V(D) = \{v_2, v_3\}$



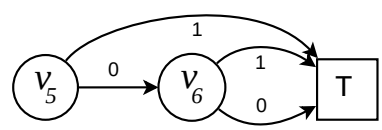
Step VII: decompose(G_8)
$G_8 = C$ (Fig. 1.1), where $V(C) = \{v_3, v_4\}$



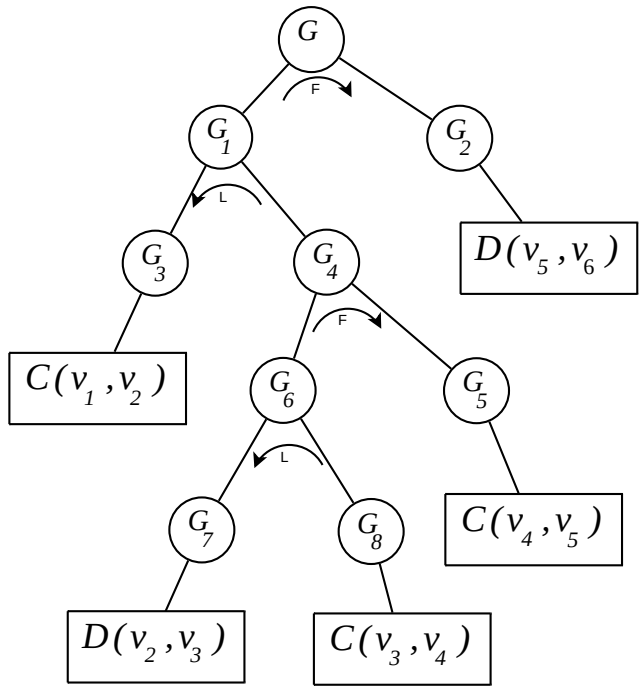
Step VIII: decompose(G_5)
$G_5 = C$ (Fig. 1.1), where $V(C) = \{v_4, v_5\}$



Step IX: decompose(G_1)
$G_1 = D$ (Fig. 1.1), where $V(D) = \{v_5, v_6\}$



Node w in Algorithm 2 is, according to the proof of Lemma 6.1, the first or the last internal node of the graph H. We keep the information in the internal nodes of the recursion tree using the labels F (first) and L (last). Hence the initial graph can be formed by the following recursion tree:



To generate the graph $G \in SPG$, we traverse the recursion tree in postorder (first the subtrees, then the root). In every node S, which has 2 subtrees, we do the following:

- if the label is L, then we execute the superposition $S = H_{w \leftarrow F}$, where H is the left subtree, F is the right subtree and w is the last node in Hamiltonian path of the graph H;
- if the label is F, then we execute the superposition $S = H_{w \leftarrow F}$, where H is the right subtree, F is the left subtree and w is the first node in Hamiltonian path of the graph H.

CONCLUSION

The original aim of this work was to develop necessary and sufficient properties to describe the superpositional graphs without using the superposition. The development of the methodology presented in this thesis was prompted by the fact that a description of SPGs with *classical properties* could not be found, the set of the properties found turned out to be incomplete.

To find similar structures, we counted SPGs manually. Based on the results, we found the problem (the bracketing problem) from The Encyclopedia of Integer Sequences that was similar to the task of finding the numbers of n-node SPGs. The integer sequence found there provided guidance for the next task. Our goal was to find such a family of propositional formulae, which depends on the parameter n and which generates large Schröder numbers.

In this thesis we found the descriptive properties for the class of superpositional graphs by using a new approximation methodology. The idea of this methodology is to effectively find models from current approximation of description that do not fit with the structure description. This is done with the help of a special translator and a counter.

By using the translator to find the classical propositional formulae from parameterized propositional formulae, we found subsequent sets of properties that generate integer sequences, which are gradually closer to the correct integer sequences. By examining the final set, we made sure that we cannot leave out any of the properties. Thereafter, we proved that the last set of properties describes exactly the class of SPGs and we got answers to the problems raised.

We conclude that the described methodology and tools (the translator and the counter of satisfying assignments) allows us to effectively compare sets of properties, i.e., if one subsumes the other, the second one first or none of them holds (for example, Chapter 5, Approximation IV). Thereby, the methodology allows to effectively detect tautologies and find models that do not fit in the structure description.

We estimate that the created methodology is useful for solving other problems.

BIBLIOGRAPHY

[1] Akers, S. B. Binary decision diagram. *IEEE Trans. on Computers*, **C-27**, 6, (1978), 509–516.

[2] Bryant, R. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, **C-35** (1986), 677–691.

[3] Drechsler, R., Becker, B. Binary Decision Diagrams, Theory and Implementation. Kluwer Academic Publishers (1998), 200 pp.

[4] Jutman, A., Peder, A., Raik, J., Tombak, M., Ubar, R. Structurally synthesized binary decision diagrams. In *Proceedings of the 6th International Workshop on Boolean Problems*, Freiberg University (2004), 271–278.

[5] Jutman, A., Raik, J., Ubar, R. SSBDD model : advantageous properties and efficient simulation algorithms. In *Proceedings of the 7th IEEE European Test Workshop*, [S.l.], (2002), 345–346.

[6] Lee, C. Representation of Switching Circuits by Binary Decision Diagrams. *Bell. Syst. Tech. Journal*, **38** (1959), 985–999.

[7] Minato, S. Binary Decision Diagrams and Applications for VLSI CAD. Kluwer Academic Publishers (1996), 141 pp.

[8] Peder, A., Isotamm, A., Tombak, M. A Meta-compiler for propositional formulae. In *Proceedings of the Seventh Symposium on Programming Languages and Software Tools*, Szeged, Hungary (2001), 250–261.

[9] Peder, A., Tombak, M. Superpositional graphs. *Acta et Commentationes Universitatis Tartuensis de Mathematica*, **13**, (2009), 51–64.

[10] Sloane, N. On-Line Encyclopedia of Integer Sequences. `http://www.research.att.com/~njas/sequences/`

[11] Sloane, N., Plouffe, S. The Encyclopedia of Integer Sequences. Academic Press, San Diego (1995), 587 pp.

[12] Stamm-Wilbrandt, H. Programming in Propositional Logic or Reductions: Back to the Roots (Satisfiability). In *Technical reports of Department of Computer Science III*, University of Bonn (1993), 24 pp.

[13] Stanley, R.P. Hipparchus, Plutarch, Schröder and Hough. *Am. Math. Monthly*, **104** (1997), 4, 344–350.

[14] Tombak, M., Isotamm, A., Tamme, T. On Logical method for Counting Dedekind Numbers. In *Proceedings of the FCT'2001*, Riga, LNCS **2138** (2001), 424–427.

[15] Tombak, M., Sillitoe, I. On the Superposition of Boolean Functions. *Annales Universitatis Scientarum Budapestinensis de Rolando Etvs nominatae, sectio computatorica*, **tomus XVII** (1998), 381–389.

[16] Ubar, R. Research and Development of Testing Methods for Digital Systems. DSc Dissertation, Institute of Electronics and Computer Science, Riga (1986), 496 pp.

[17] Ubar, R. Test Generation for Digital Circuits Using Alternative Graphs (in Russian). In *Proc. Tallinn Technical University*, **409**, Tallinn TU, Tallinn, Estonia (1976), 75–81.

[18] Ubar, R., Devadze, S., Raik, J. Ultra Fast Parallel Fault Analysis on Structural BDDs. In *Proceedings of the 12th IEEE European Test Symposium*, IEEE press, (2007), 131–136.

[19] Ubar, R., Devadze, S., Raik, J., Jutman, A. Parallel X-fault Simulation with Critical Path Tracing Technique. In *Proc. of the IEEE/ACM DATE conference*, Dresden, Germany, 2010, 6 pp.

[20] Ubar, R., Vassiljeva, T., Raik, J., Jutman, A., Tombak, M., Peder, A. Optimization of structurally synthesized BDDs. In *Proceedings of the Fourth IASTED International Conference on Modelling, Simulation and Optimization*, Kauai, HI, Acta Press (2004), 234–240.

[21] Vlach, F. Simplification in a Satisfiability Checker for VLSI Applications. *Journal of Automated Reasoning*, **10** (1993), 115–136.

[22] Weisstein, Eric W. Schröder Number. Wolfram MathWorld. `http://mathworld.wolfram.com/SchroederNumber.html`

[23] Satisfiability Suggested Format. Center for Discrete Mathematics & Theoretical Computer Science (DIMACS) (1993), 8 pp.

[24] The First DIMACS International Algorithm Implementation Challenge: Problem Definitions and Specifications. Center for Discrete Mathematics & Theoretical Computer Science (DIMACS) (1999), 13 pp.

[25] Peder, A. A Language and Compiler for Families of Propositional Formulae. `http://www.ut.ee/~ahtip/LOG_COMP/main.html`

SUPERPOSITSIOONIGRAAFID JA STRUKTUURI KIRJELDUSE LEIDMINE LOENDAMISMEETODI ABIL

Käesolevas doktoritöös käsitletav temaatika sai alguse struktuurselt sünteesitud binaarsete otsustusdiagrammide (*Structurally Synthesized Binary Decision Diagrams*, SSBDD) uurimisest. Olulisim erinevus traditsioonilise BDD [2] ja SSBDD vahel on nende genereerimise viisis. Kui BDD on saadud Shannoni lahutuse teel, siis SSBDD jaoks on kasutatud spetsiaalset protseduuri – superpositsiooni. Esimest korda on neid graafe käsitlenud Raimund Ubar aastal 1976 artiklis [17] nime all struktuursed alternatiivsed graafid. Kuna paljud SSBDD omadused sõltuvad tema "skeleti", superpositsioonigraafi (SPG), vastavatest omadustest [4], siis oli eesmärk töötada välja SPG-de klassi kirjeldavad omadused. Seega oli vaja lahendada järgmised probleemid:

- millistel tingimustel binaarne graaf G on SPG;
- kuidas $G \in SPG$ jaoks leida efektiivselt sellist superpositsioonide jada, mis seda genereerib.

Kuna SPG kui andmestruktuur ei sisalda endas saamise ajalugu (milliseid superpositsioone selle saamiseks rakendati), siis ei tohi otsitavad SPG-de klassi kirjeldavad omadused superpositsiooni kasutada. Kui õnnestub leida tarvilikud ja piisavad tingimused traditsioonilise graafiteooria terminites superpositsiooni kasutamata, siis loodetavasti võimaldab süvendatud arusaam SPG-de olemusest leida kiired algoritmid ülalnimetatud kahe probleemi lahendamiseks.

Käesolevas töös esitletud metoodika arendus sai alguse asjaolust, et SPG-de kirjeldust *tavaliste* omadustega ei õnnestunud leida. Leitud omaduste hulk osutus mittetäielikuks.

Et leida sarnaseid struktuure, loendasime SPG-sid *jõumeetodil*. Saadud tulemusest lähtuvalt leidsime arvjadade entsüklopeedia [10] abil probleemi (sulgavaldiste probleem, [13]), mis on isomorfne meie ülesandega leida n-tipuliste SPG-de arv.

Seda iseloomustav arvjada sai järgneva töö orientiiriks. Eesmärk oli leida selline ühest parameetrist (n – SPG tippude arv) sõltuv lausearvutusvalemite pere $\mathcal{F}_n$, mille korral loendamisülesanne $\#SAT(\mathcal{F}_n)$ annab parameetri n väärtuste $1, 2, 3, \ldots$ puhul vastuseks suured Schröderi arvud

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, \ldots$$

Käesolevas töös on leitud SPG-de klassi kirjeldavad omadused uue lähendamismeetodiga. Võtmetähtsusega on seejuures translaator [8], mis leiab etteantud perele ja selle parameetrite väärtustele vastava lausearvutusvalemi.

Translaatorit ja kehtestavate väärtustuste loendajat kasutades leiame sammsammult õigemat arvjada genereeriva omaduste komplekti. Metoodika põhiidee on leida efektiivselt mudeleid, mis ei sobi struktuuri kirjeldusega, ning seejärel parandada lähendiks olevat omaduste komplekti. Kui nähtub, et leitud omaduste komplekt genereerib arvjada, mis langeb piisavalt suurte parameetri väärtuste puhul kokku orientiiriks võetud arvjadaga, siis saame püstitada hüpoteesi, et viimasena leitud lähend ongi otsitav. Seejärel tõestame hüpoteesi. Translaatori ja loendaja abil saab ka veenduda, et ükski omadus ei ole ülearune.

Käesoleva doktoritöö põhitulemused on:

- metoodika väljatöötamine hüpoteeside püstitamiseks loendamismeetodil;
- valemiperede kirjeldamiseks vajaliku grammatika ja translaatori loomine, saamaks efektiivselt valemiperest parameetriteta lausearvutusvalem;
- superpositsioonigraafide klassi kirjelduse leidmine.

Kirjeldatud metoodika võimaldab tõhusalt leida mudeleid mis ei sobi struktuuri kirjeldusega. Ka võimaldab metoodika efektiivselt kontrollida, kas struktuuri kirjeldavate omaduste hulgas on ülearuseid s.t. leida tautoloogiaid (näiteks peatükk 5, lähend IV). Väljendades omaduste hulki lausearvutusvalemite peredena, saame arvjadade entsüklopeediast lihtsasti leida sarnaseid struktuure. Nende leidmine aga vastava arvjadata võib osutuda keeruliseks.

Arvatavasti saab loodud metoodikat rakendada ka teiste probleemide lahendamiseks.

CURRICULUM VITAE

Name: Ahti Peder
Citizenship: Estonia
Born: August 29, 1976, Tarvastu, Estonia
Marital Status: common-law marriage, 2 children
Contacts: (+372) 375482, Ahti.Peder@ut.ee

Education

1995–1999: Faculty of Mathematics, University of Tartu, BSc in informatics

1999–2001: Faculty of Mathematics, University of Tartu, MSc in informatics

2001–. . . **:** Faculty of Mathematics, University of Tartu, PhD studies in informatics

Professional employment

2001 - . . . **:** Assistant, Institute of Computer Science, University of Tartu

ELULOOKIRJELDUS

Nimi: Ahti Peder
Kodakondsus: Eesti
Sünniaeg: 29. august 1976, Tarvastu, Viljandimaa
Perekonnaseis: vabaabielus, 2 last
Kontaktandmed: (+372) 375482, Ahti.Peder@ut.ee

Haridus
1995–1999: Tartu Ülikooli matemaatikateaduskond, BSc informaatikas
1999–2001: Tartu Ülikooli matemaatikateaduskond, MSc informaatikas
2001–. . . **:** Tartu Ülikooli matemaatikateaduskond, doktoriõpe infomaatikas

Erialane teenistuskäik
2001 - . . . **:** Tartu Ülikooli arvutiteaduse instituudi assistent

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. Mati Heinloo. The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu 1991, 23 p.
2. Boris Komrakov. Primitive actions and the Sophus Lie problem. Tartu 1991, 14 p.
3. Jaak Heinloo. Phenomenological (continuum) theory of turbulence. Tartu 1992, 47 p.
4. Ants Tauts. Infinite formulae in intuitionistic logic of higher order. Tartu 1992, 15 p.
5. Tarmo Soomere. Kinetic theory of Rossby waves. Tartu 1992, 32 p.
6. Jüri Majak. Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu 1992, 32 p.
7. Ants Aasma. Matrix transformations of summability and absolute summability fields of matrix methods. Tartu 1993, 17 p.
8. Helle Hein. Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu 1993, 29 p.
9. Toomas Kiho. Study of optimality of iterated Lavrentiev method and its generalizations. Tartu 1994, 24 p.
10. Arne Kokk. Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu 1995, 166 p.
11. Тоомас Лепикулт. Автоматизированный расчет задач динамики жесткопластических конструкций. Тарту 1995, 88 с.
12. Сандер Ханнус. Параметрическая оптимизация пластических цилиндрических оболочек с учетом геометрической и физической нелинейности. Тарту 1995, 75 с.
13. Sergei Tupailo. Hibert's epsilon-symbol in predicative subsystems of analysis. Tartu 1996, 132 p.
14. Enno Saks. Analysis and optimization of elastic-plastic shafts in torsion. Tartu 1996, 94 p.
15. Valdis Laan. Pullbacks and flatness properties of acts. Tartu 1999, 88 p.

16. Märt Põldvere. Subspaces of Banach spaces having Phelps' uniqueness property. Tartu 1999, 72 p.
17. Jelena Ausekle. Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu 1999, 70 p.
18. Krista Fischer. Structural mean models for analyzing the effect of compliance in clinical trials. Tartu 1999, 124 p.
19. Helger Lipmaa. Secure and efficient time-stamping systems. Tartu 1999, 56 p.
20. Jüri Lember. Consistency of empirical k-centres. Tartu 1999, 146 p.
21. Ella Puman. Optimization of plastic conical shells. Tartu 2000, 102 p.
22. Kaili Müürisep. Eesti keele arvutigrammatika: süntaks. Tartu 2000, 107 lk.
23. Varmo Vene. Categorical programming with inductive and conductive types. Tartu 2000, 116 p.
24. Olga Sokratova. Ω-rings, their flat and projective acts with some applications. Tartu 2000, 121 p.
25. Maria Zeltser. Investigation of double sequence spaces by soft and hard analytical methods. Tartu 2001, 153 p.
26. Ernst Tungel. Optimization of plastic spherical shells. Tartu 2001, 90 p.
27. Tiina Puolakainen. Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu 2001, 137 lk.
28. Rainis Haller. $M(r, s)$-inequalities. Tartu 2002, 77 p.
29. Jan Villemson. Size-efficient interval time stamps. Tartu 2002, 81 p.
30. Eno Tõnisson. Solving of expression manipulation exercises in computer algebra systems. Tartu 2002, 91 p.
31. Mart Abel. Structure of Gelfand-Mazur algebras. Tartu 2003, 92 p.
32. Vladimir Kuchmei. Affine completeness of some Ockham algebras. Tartu 2003, 99 p.
33. Olga Dunajeva. Asymptotic matrix methods in statistical inference problems. Tartu 2003, 77 p.
34. Mare Tarang. Stability of the spline collocation method for Volterra integro-differential equations. Tartu 2004, 88 p.
35. Tatjana Nahtman. Permutation invariance and reparameterizations in linear models. Tartu 2004, 89 p.
36. Märt Möls. Linear mixed models with equivalent predictors. Tartu 2004, 69 p.
37. Kristiina Hakk. Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 136 p.
38. Meelis Käärik. Fitting Sets to Probability Distributions. Tartu 2005, 89 p.
39. Inga Parts. Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.

40. Natalia Saealle. Convergence and summability with speed of functional series. Tartu 2005, 90 p.
41. Tanel Kaart. The reliability of linear mixed models in genetic studies. Tartu 2006, 123 p.
42. Kadre Torn. Shear and bending response of inelastic structures to dynamic loads. Tartu 2006, 141 p.
43. Kristel Mikkor. Uniform factorization for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. Darja Saveljeva. Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 116 p.
45. Kristo Heero. Path Planning and Learning Strategies for Mobile Robots in Dynamic Partially Unknown Environments. Tartu 2006, 122 p.
46. Annely Mürk. Optimization of inelastic plates with cracks. Tartu 2006, 136 p.
47. Annemai Raidjõe. Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. Olga Panova. Real Gelfand-Mazur algebras. Tartu 2006, 81 p.
49. Härmel Nestra. Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 118 p.
50. Margus Pihlak. Approximation of multivariate distribution functions. Tartu 2007, 81 p.
51. Ene Käärik. Handling dropouts in repeated measurements using copulas. Tartu 2007, 98 p.
52. Artur Sepp. Affine models in mathematical finance: an analytical approach. Tartu 2007, 148 p.
53. Marina Issakova. Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 169 p.
54. Kaja Sõstra. Restriction estimator for domains. Tartu 2007, 103 p.
55. Kaarel Kaljurand. Attempto Controlled English as a Semantic Web Language. Tartu 2008, 161 p.
56. Mart Anton. Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. Evely Leetma. Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. Ants Kaasik. Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 136 p.
59. Reimo Palm. Numerical comparison of regularization algorithms for solving ill-posed problems. Tartu 2010, 105 p.
60. Indrek Zolk. The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. Jüri Reimand. Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.