Application of Hypergraphs in Decomposition of Discrete Systems

Faculty of Electrical Engineering, Computer Science and Telecommunications
University of Zielona Góra

Lecture Notes in Control and Computer Science Volume 23

Monika Wiśniewska

Application of Hypergraphs in Decomposition of Discrete Systems

University of Zielona Góra Press, Poland
2012

Monika WIŚNIEWSKA
Institute of Computer Engineering and Electronics
University of Zielona Góra
ul. Podgórna 50
65-246 Zielona Góra, Poland
e-mail: M.Wisniewska@weit.uz.zgora.pl

Supervisor:

- Marian ADAMSKI, University of Zielona Góra

Referees:

- Larisa TITARENKO, University of Zielona Góra
- Dariusz KANIA, Silesian University of Technology

The text of this book was prepared based on the author's Ph.D. dissertation entitled *Dekompozycja systemów dyskretnych z wykorzystaniem hipergrafów*

ISBN 978-83-7842-025-5

Camera-ready copy prepared in LaTeX2$_\varepsilon$ by the author

University of Zielona Góra Press
ul. Licealna 9, 65-417 Zielona Góra, Poland
tel./fax: +48 68 328 78 64, e-mail: oficynawydawnicza@adm.uz.zgora.pl

Printed by the University of Zielona Góra Printing House

Contents

List of Symbols and Notations

H - hypergraph H

G - graph G

A - incidence matrix of hypergraph H (or graph G)

V - the set of vertices of hypergraph H (or graph G); V=$\{v_1, \ldots, v_n\}$

n - the number of vertices of hypergraph H (or graph G)

E - the set of edges of hypergraph H (or graph G); E=$\{E_1, \ldots, E_m\}$

m - the number of edges of hypergraph H (or graph G)

H^* - hypergraph dual to H

$\overline{H}$ - complement of hypergraph H

$\overline{G}$ - complement of graph G

Cl - clique Cl (in graph G)

P - subgraph P (of graph G)

K_n - complete graph (containing n vertices)

T - transversal (hitting set) of a hypergraph H

τ - the minimal transversal of hypergraph H

D - the exact transversal of hypergraph H

δ - the minimal exact transversal of hypergraph H

k - number of all colors of hypergraph H (or graph G)

$\chi(H)$ - chromatic number of hypergraph H

$\chi(G)$ - chromatic number of graph G

C - compatibility class (independent set) of hypergraph H (or graph G)

L_i - weight (cost) of the i-th compatibility class C_i

C_C - the set of compatibility classes of hypergraph H (or graph G); C_C=$\{C_1, \ldots, C_K\}$

PN - Petri net PN

P - the set of places of a Petri net; P=$\{P_1, \ldots, P_{|P|}\}$

P_i - single (i-th) place of Petri net PN

T - the set of transitions zbiór of a Petri net; T=$\{T_1, \ldots, T_{|T|}\}$

T_i - single (i-th) transition of Petri net PN

MP - the set of macroplaces of a Petri net

M_i $(or\ MP_i)$ - single (i-th) macroplace of Petri net PN

X - the set of input signals of a discrete system; X=$\{x_1, \ldots, x_{|X|}\}$

Y - the set of output signals of a discrete system (the set of microoperations); Y=$\{y_1, \ldots, y_N\}$

Q - the set of variables for encoding of microoperations; Q=$\{q_1, \ldots, q_{|Q|}\}$

μ - the set of microinstructions; μ=$\{\mu_1, \ldots, \mu_M\}$

Chapter 1

INTRODUCTION

Recent years have witnessed the increasing impact of the theory of graphs on the development of digital technology. This is due to the fact that logic systems have grown in size and, what follows, innovative designing methods for digital systems have been developed. Yet a few years ago designers could specify and elaborate the functionality of their systems easily and quickly, while today it is impossible without the application of computer-aided design tools. Numerous current decomposition methods for discrete systems, used by leading manufacturers who prepare tools for the synthesis of digital systems, rely on the utilisation of classical undirected graphs. Such an approach exerts the application of ever more advanced computational methods as well as the modification of the existing algorithms. It is caused by the continuous growth of the size of constructed digital systems, what in turn results in essential changes in the implementation of the developed model in a real logic system.

One of the possible solutions to the problem may be the application of hypergraph theory and related algorithms. The application of hypergraphs to the operations used in digital technology, such as, e.g., minimization of logical functions (DeMicheli, 1994; Eiter, 1994; Łuba, 2005) or partitioning of a discrete system (Lee-Kwang and Cho, 1996; Leinweber and Bhunia, 2008), seems to be intuitive, more transparent and more effective than in the case of classical graphs (Wiśniewska and Adamski, 2008b).

This book includes the analysis of some of the existing general methods which support the design of digital systems with the use of hypergraphs, whose effectiveness and the operation speed were thoroughly examined. On this basis the author's own new algorithms, binding the theory of hypergraphs with the theory of Petri nets, were developed. The algorithms are to solve a given problem faster and more efficiently than the so far applied methods. Moreover, an innovative application supporting digital system design with the use of the theory of hypergraphs was developed and implemented.

1.1. Current State of Knowledge, Motivation for Addressing the Issue

The work considers discrete systems described with the use of Petri nets (Murata, 1989; Girault and Valk, 2003) or equivalent transition systems. The examined

space of discrete states consists of fundamental local states grouped in global states. Local states refer to the places in a Petri net, whereas global states refer to the respective states of a given transition system (Adamski, 1990; Węgrzyn *et al.*, 1996; Karatkevich, 2007; Zakrevskii, 1986). The scope of issues considered in this book is limited to the analysis of such discrete systems in which the states of a transition system correspond to the global states of a Petri net. Such a restraint is a generally assumed practice, used in the analysis of Petri net state spaces, places and transitions (PT-nets), and discussed, among others, in the works of (David and Alla, 1992; Banaszak *et al.*, 2008). The subject of the research is the analysis of the concurrency relation between local states, in terms of the decomposition of a discrete system into concurrent subsystems. The assumed restriction, resulting from theoretical and practical reasons, is the postulate that all local states in a concurrent subsystem remain in a sequentiality relation.

The analysis of discrete systems described with Petri nets in terms of decomposition has been presented in numerous publications. M. Adamski recapitulates the results of the research carried out with the use of an original method of the analysis of undirected concurrency graphs (Adamski, 1990). Parallel works of a similar range were carried out in the National Academy of Sciences of Belarus (A. Zakrevskij), University of Bristol (E. Dagless, M. Bolton) and University of Minho in Braga and Guimaraes (J.L. Monteiro, A.J. Proença). The work of A. Karatkevich, (Karatkevich, 2007) constitutes a valuable monograph on various aspects of dynamic analysis of Petri-net based discrete systems.

The review of the literature reveals that the proposed in Zielona Góra concept of the application of hypergraphs in the representation of Petri net space states has an innovative character on a worldwide scale. The fundamental trend of the research was realized by the author of this book.

The discussed hypergraphs, unlike ordinary undirected graphs, present the relations between both local and global states of an analyzed discrete system. Owing to this fact, the decomposition methods for a discrete system become essentially simpler, since the data for the analysis consists of the spaces of both local and global states. In consequence, it leads to a faster process of obtaining the results, especially for the Petri net state spaces described by exact hypergraphs.

1.2. The Thesis, Objectives and Tasks of the Work

On the basis of the conducted research as well as the review of the world literature, the following thesis of the work was formulated: *The application of hypergraphs into the decomposition of state space in discrete systems improves the process of designing concurrent digital automata owing to effective mathematical methods for examining the graph structures.*

The term effectiveness of the computational method refers to its efficiency (obtaining correct results) as well as to its performance (obtaining the result in an acceptable time) with the use of available computer-aided tools.

The main thesis implies the following detailed assumptions:

- an exact hypergraph unambiguously represents the local state space of Petri nets;
- a state machine component (SMC) of a regular Petri net corresponds to an exact transversal of a concurrency hypergraph.

These terms are discussed in details in Chapter 5. The objective of the work is the proposal, development and algorithmisation of decomposition methods for discrete systems, useful in the design of configurable logic controllers, of considerable practical values and computational efficiency.

The main tasks of the book are:

1. Improvement of the decomposition methods for discrete systems by the use of the theory of hypergraphs, which is intensively and independently developed and turned into algorithms for the needs of computational logic.
2. Development of innovative, more efficient decomposition methods for discrete systems with the application of hypergraphs.
3. Realization of a computer aided design system for logic controllers with the application of hypergraph algorithms (CAD).

1.3. The Structure of the Book

The work consists of eight chapters and three appendices. The first chapter includes the introduction to the ideas covered by the book, formulation of the thesis as well as the objectives and tasks of the book.

Chapter Two deals with discrete systems and possible ways of their representation. Most important ideas relating to a discrete system described with Petri nets are discussed, with a particular emphasis to the realization of the structures in the form of concurrent digital automata.

The basic concepts of the theory of undirected graphs and hypergraphs are discussed in Chapter Three, whereas the most important algorithms are addressed in Chapter Four. Mathematical research methods in graphs and hypergraphs, such as the determination of a complement, vertex cover, coloring, or the determination of exact transversals, are thoroughly discussed. All the algorithms were implemented in the author's own *Hippo* system, which is widely described in Appendix A.

Chapters Five and Six describe innovative solutions introduced in the book. Chapter Five includes the author's definitions and theses relating to the decomposition of discrete systems. Chapter Six presents the author's decomposition methods for discrete systems.

The verification of the effectiveness of the developed methods is carried out in Chapter Seven. It presents the author's own tools supporting the research process, together with the libraries for test modules, and the results of the experiments.

Chapter Eight includes the summary and conclusions. What is more, it outlines possible directions of further research.

Appendix A includes the description of particular modules of the author's *Hippo* system, which supports the decomposition of discrete systems with the application of hypergraphs. Appendix B contains a detailed list of the results of the experimental research. Appendix C contains Verilog source codes which illustrate a possible way to realize exemplary discrete systems presented in the book.

Chapter 2

DISCRETE SYSTEM

The Chapter presents basic problems connected with discrete systems as well as the methods of their representation. It also introduces the most important definitions and relations (concurrency, sequentiality) between the states of a discrete system.

2.1. The Representation of a Discrete System

Definition 2.1. ***A discrete system*** *is a system consisting of a finite number of local and global states (Steiglitz, 1974; Adamski, 1990; Banaszak et al., 2008). In a discrete system, one or more local states can be active simultaneously. Global states are described by maximal sets of all active local states. Since discrete systems considered in the book may belong to a finite number of the reached global states, they may be precisely illustrated by finite graph representations and mathematical models (Wiśniewska et al., 2007a).*

Discrete systems model arbitrary discrete processes, both sequential and concurrent. They are very frequently applied to model and represent controlling systems described by Petri nets. The successive subchapters include the discussion on the selected methods of the representation of Petri nets used in the work.

2.1.1. A Petri Net

Definition 2.2. ***A Petri net*** *is a directed bipartite graph with two kinds of vertices: places and transitions connected by a directed arc (Petri, 1962; Murata, 1989; Banaszak et al., 1993; Girault and Valk, 2003). A Petri net is defined by the following 3-tuple:*

$$PN = (P, T, F), \tag{2.1}$$

where:
P *is a finite, nonempty set of places;*
T *is a finite nonempty set of transitions;*
$F \subseteq (P \times T) \bigcup (T \times P)$*, is a finite, nonempty set of arcs.*

Petri nets are used to represent the discrete systems in which mutually conditioning states and events occur. Places are usually interpreted as conditions (local states of a discrete system), whereas transitions - as events (alterations of local states). The conditions imposed on an incidence relation result in the fact that

each place and each transition should be associated with an arc. It means that, in the considered interpretation, each event (transition) responds to a certain set of places (conditions) enabling an event, and consequently satisfying a set of output conditions, represented by suitable output places of the transition. Satisfying the output conditions may thus mean satisfying successive input conditions, etc. Transitions are *active*, if in all input transition places there is at least one token, whereas the firing of a transition is understood as the removal of a single token from input places and placing it in transition output places.

Let's introduce some definitions that will be used further in the book to describe the properties of a Petri net.

Definition 2.3. *A **state machine (SM)** is a subclass of a Petri net such that each transition t has exactly one input place and one output place (Murata, 1989):*

$$\forall t \in T : |\bullet\, t| = 1 = |t \,\bullet|. \tag{2.2}$$

Definition 2.4. *A **state machine component (SM-component, SMC)** of a Petri net PN is a strongly connected subnet PN' generated by places in PN (Murata, 1989):*

- *each transition of a subnet PN' has exactly one input and one output arc;*
- *all input and output transitions of places in PN' and their connecting arcs belong to PN'.*

Definition 2.5. *A **marked graph (MG)** is a subclass of a Petri net such as each place has exactly one input transition and exactly one output transition (Murata, 1989):*

$$\forall p \in P : |\bullet\, p| = 1 = |p \,\bullet|. \tag{2.3}$$

Each Petri net has its own behavioral properties (Murata, 1989). The work presents the selected properties of Petri nets which are used in the further part of the book, for the description of decomposition algorithms of discrete systems with the application of hypergraphs:

- ⋆ *Reachability.* A marking M_n is said to be reachable from the initial marking M_0 if there is a sequence of firings that transforms M_0 to M_n.
- ⋆ *Liveness.* The transition t is live if from any marking M_n of a Petri net it is possible to fire transition t by a sequence of firings of other transitions. A Petri net is live if from any marking M_n of a Petri net it is possible to fire any transition in a net by a sequence of firings of other transitions.
- ⋆ *Safeness.* A place p of a Petri net is safe if there is no reachable state that contains more than one token in this place. A Petri net is safe if each place in the net is safe. Thus, places of a safe net model Boolean conditions, which may be either satisfied (token present) or not (token absent). The *liveness* and the *safety* of a net may be tested by determining a reachability graph, which represents the tree where repetitive markings have been combined in a single node. Problems connected with reachability of particular states in a Petri net are described in details in the further part of the work.

The book considers live and safe Petri nets that belong to the subclass of marked graphs (Murata, 1989). The decompositions of Petri nets into state machine components (subnets of an automaton type) are discussed in details in Chapter 6.

2.1.2. An Interpreted Petri Net

Definition 2.6. ***An interpreted (labeled) Petri net*** *is a live and safe Petri net with defined input and output signals (Girault and Valk, 2003). Both sets of inputs X and outputs Y are defined by binary vectors:*
$X = \{x_1, x_2, \ldots, x_n\}$,
$Y = \{y_1, y_2, \ldots, y_m\}$.

An interpreted Petri net is a refinement of Definition 2.2 (Banaszak *et al.*, 1993). The consideration of binary vectors of input and output states enables a very convenient representation of binary systems (logic systems) which may be realised e.g., using digital systems. A concurrent digital automaton (more extensively presented in Chapter 2.2) is most frequently used as an implementation model. The automaton outputs may be defined as binary outputs of Moore's type (if the output state depends on the configuration of active local states exclusively) or of Mealy's type (when the output state is also directly conditioned by the output state of the system).

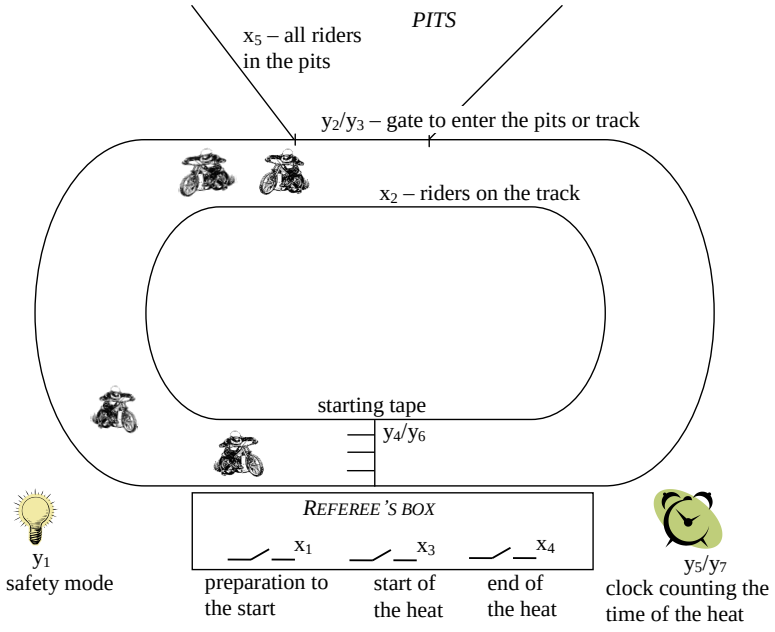

Fig. 2.1. Simplified controlling system of a speedway tournament

To shed some light on the issue, a real-life problem will be presented as an example illustrating how Petri nets are used to represent discrete processes. Figure 2.1 presents a simplified system controlling the preparation to a single heat and race itself during a speedway tournament. Initially, the system is in the standby mode to begin preparations to a heat. After the referee has pushed button x_1, three concurrent operations are performed. A pit gate is opened and the riders are allowed to get on track (which is illustrated by active output signal y_2). At the same time, a starting tape is being prepared (a "ready" position of the tape is illustrated by active signal y_4). Simultaneously, a safety mode is turned on, which means that unauthorised persons are not permitted on to the track (only the riders are allowed to be on the track). The mode is induced by active output signal y_1.

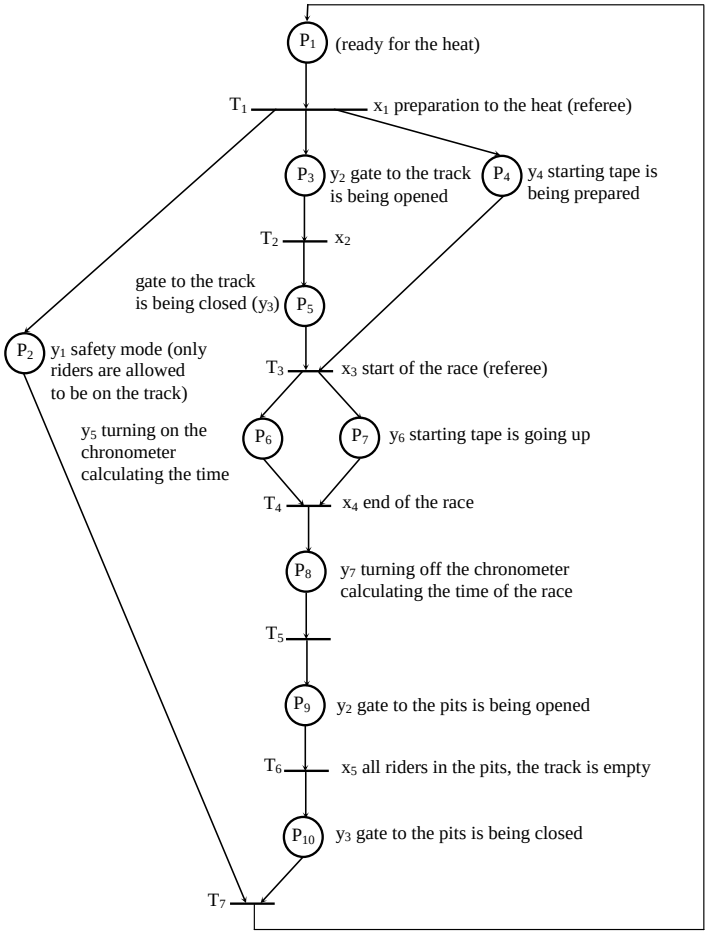

Fig. 2.2. Interpreted Petri net PN_1

When all the riders racing in a heat enter the track (which is signalled with x_2), the pit gate is closed (y_3). Waiting for the heat to start is the subsequent phase. Signal x_3 indicates that the starting button has been pushed by the race referee, which results in turning on a chronometer calculating the time of the race, (signal y_5) and raising the starting tape (y_6). The finish of the race is signalled with condition x_4. The chronometer is turned off, next the pit gate is opened. When all the riders leave the track and get to the pits (active condition x_5), the gate is closed and next the safety mode is turned on. The procedure of a single heat is finished, and the system goes to the standby mode, waiting for the next race.

The system was described by Petri net PN_1 shown in Fig. 2.2. The presented controller, from the formal point of view, is a digital automaton with five input signals, seven output signals and ten local states.

2.1.2.1. A Macronet

For complex control systems described with Petri nets, the reduction methods which allow simplification of the initial structure are frequently used. For this purpose, the feature of hierarchy of Petri nets can be exploited to create *a macronet*. This means that fragments of the initial Petri net are replaced with *macroplaces*. The operation may simplify considerably the analysis (or decomposition) of a given net, not affecting the final result.

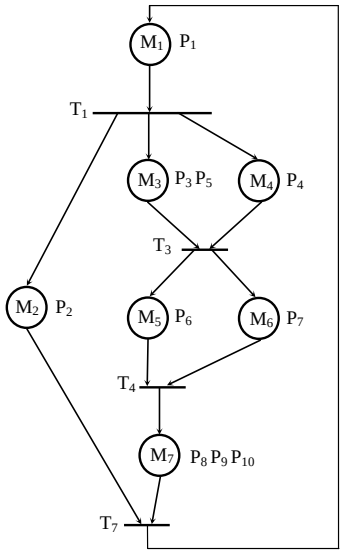

Fig. 2.3. Macronet MN_1 for Petri net PN_1

The determination of a macronet is carried out with the "descending structural method" (from generalities to details), obtaining a base net and a set of mutually nesting subnets (Banaszak *et al.*, 1993). A detailed description of the reduction of Petri nets and the determination of macronets can be found in the literature (Peterson, 1981; Murata, 1989; Karatkevich, 2008). Figure 2.3 presents macronet MN_1 for Petri net PN_1. The sequential fragments of the net were replaced with macroplaces. The resultant macronet MN_1 is a condensed version of net PN_1, maintaining its features and properties. The reduction allowed the significant decrease of the initial form of the Petri net. Structure PN_1 consists of 10 places, whereas its reduced equivalent includes 7 macroplaces (noted with symbol M, other frequently used symbol is MP). The figure demonstrates how places of initial net PN_1 form sequential macroplaces in macronet MN_1 (e.g., macroplace M_7 includes places P_8, P_9, P_{10}).

2.1.3. A Transition System

Petri nets were developed primarily to analyse sequential and concurrent processes which take place in a designed discrete system. Most frequently, the first step in the examination of Petri net properties (liveness, safety) is to determine *the reachability graph* (other names - *a marking graph* or *a graph of reachable markings*).

The structure is determined on the basis of the initial Petri net or its condensed version, i.e. a macronet (the representation of a Petri net by a macronet allows shortening the concurrency analysis, which was shown in (Kovalyov, 1992; Adamski *et al.*, 2005; Karatkevich, 2007)). The formation process of a reachability graph is by the analysis of alterations in net marking when the transitions are ready to be fired. The values are described in the form of a graph whose vertices respond to a set of places marked in a given state, whereas its edges determine the fired transitions (Murata, 1989; Adamski and Chodań, 2000; Karatkevich, 2008).

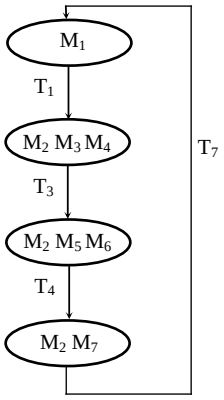

Fig. 2.4. Marking graph for macronet MN_1

Figure 2.4 presents a graph of markings for macronet MN_1. The structure illustrates basic relations which occur in the considered automaton. As it results from the figure, places M_2, M_3 and M_4 are marked in the same macrostate. It means that the three fragments are performed concurrently (Murata, 1989; Roguska, 2001).

A reachability graph of a Petri net is *a transition system*, where global states (vertices of a marking graph) determine concurrency relations between particular local states. The system illustrates alterations in global states whose values are determined by subsequent transitions. The detailed analysis of the concurrency between particular states may be inconvenient. The next stage in the process of examining the concurrency of a Petri net may be the determination of a concurrency graph or a concurrency hypergraph which is proposed and defined in the work (Chapter 5).

2.1.4. Local States, Global States, Concurrency Relations

Discrete systems are particularly useful for describing concurrent phenomena (Banaszak *et al.*, 1993). Sequential and parallel processes may be easily presented by means of *concurrency graphs* or *hypergraphs*, which are discussed in more detail in the subsequent chapters of the book. Local states of the structures respond to places (or macroplaces) of a Petri net, whereas global states determine concurrency relations between local states.

The examination of the concurrency in discrete systems described by Petri nets is realised on the basis of *a concurrency graph* (Adamski, 1990). A concurrency graph is most frequently determined on the basis of a marking graph of a Petri net. Its vertices correspond to places (or macroplaces) of the initial Petri net, whereas the edges illustrate the concurrency between these places. If there is an edge between two places, the two fragments may be executed independently. Analogically, the lack of an edge between two vertices means that the places are mutually dependant and cannot be performed simultaneously.

Another frequently examined relation between Petri net states is a sequentiality relation. In *a sequentiality graph* (also called *a non-concurrency graph* or *a consequency graph*) there is an edge between these places of a Petri net which are not concurrent. The book considers regular Petri nets in which a sequentiality graph is a complement to a concurrency graph, and on the contrary: a concurrency graph may be determined as a complement to a sequentiality graph.

Incidence matrix $A_{G_{MN_1}}$ of a sequentiality graph, for the analysed net, is presented in Fig. 2.5. The seven vertices represent the macroplaces in net MN_1. The connections (i.e., edges in the graph) between the vertices determine the concurrency relation.

A relation between a marking graph and a concurrency graph is interesting in terms of the analysis and decomposition of a Petri net. In its structure, a marking graph stores the information about all the marked places in a given global state and connections with other global states (transitions between these states). This information disappears when a concurrency graph is determined. It describes concurrency relations between at most two local states. Concurrency determination

$$A_{G_{MN_1}} = \begin{array}{c} \begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \end{matrix}$$

Fig. 2.5. Incidence matrix of a concurrency graph for macronet MN_1

between more than two states involves the need for the determination of maximum cliques in a concurrency graph, i.e., de facto the retrieval of information which has already been stored by a marking graph. Classical decomposition methods of Petri nets are generally limited to the application of traditional methods based on undirected graphs. Therefore a notion of *a concurrency hypergraph* is defined in this book. Although the concept appeared in the author's former publications (Wiśniewska and Adamski, 2006), neither a formal definition nor the area of application were determined. A concurrency hypergraph stores all information about global states posed by a marking graph. What is more, there is no need to perform any additional transformations to determine the edges since the values are suitably stored by a marking graph. A hypergraph is defined in Chapter 5, which comprises definitions and theorems introduced in the book.

2.2. Concurrent Automata

A concurrent automaton is an abstract model of a discrete system which may be present simultaneously in one or several local states (Banaszak *et al.*, 1993). An automaton is a very useful device to describe concurrent discrete systems, which are realised with the use of digital systems. An interpreted Petri net is useful to represent a concurrent automaton (Pardey *et al.*, 1994; Kozłowski *et al.*, 1995). Local states of an automaton are reflected by the places of an interpreted Petri net, and events of a discrete system (alterations of states) are illustrated by transitions. Figure 2.6 presents a general model of a concurrent digital automaton of n binary inputs and m binary outputs.

2.2.1. A Discrete Binary System

A discrete binary system is an indirect form of the representation of a concurrent discrete system which is executed in the implementation process of a concurrent automaton in digital systems (Adamski, 1990). Such a system is formed by encoding each global state of a net with the use of binary signals. Unlike a digital

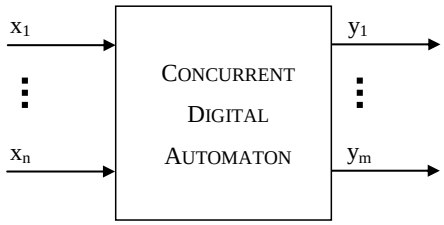

Fig. 2.6. General model of a concurrent digital automaton

automaton, a concurrent automaton may be present in more than one local state, therefore in the encoding process all concurrent relations occurring in the system must be considered.

Various encoding options are possible, depending on the assumed strategy of optimisation. The simplest method is *one-hot*), where each place of an interpreted Petri net is described with a binary value „1" (when a local state is active) or „0" (for an inactive local state). The code of a global state is determined as the concatenation of the values of all components of local states. In case of realisation of the system in digital circuits, such an approach extorts the application of a gnumber of flip-flops equal to all the local states. It is inefficient particularly in case of nets containing a relatively considerable number of places in the sequentiality relation.

Another, widely applied encoding method of the states of a concurrent automaton is a separate encoding of *state machine components* (subnets) incorporated in an initial interpreted Petri net. Each SMC forms an independent sequential system, which is formed as a result of a parallel decomposition of the interpreted Petri net (for a detailed description of a subnet as well as the whole decomposition process see Chapters 5 and 6). Internal states of each of the obtained subnets may be encoded independently, with the use of traditional encoding methods (e.g., natural binary code, Gray's code). What is more, generally known optimisation algorithms may be applied to encoding, such as Espresso method, NOVA or JEDI (Rudell, 1989; Sentovich *et al.*, 1992; Shi and Brzozowski, 1992).

The method of encoding under consideration is particularly useful for the automata realisation in PAL-type structures (Kania, 2004). The fundamental advantage of a concurrent digital automaton developed in such a way is a smaller number of flip-flops than in the case of encoding with "one-hot" method (Adamski, 1990; Banaszak *et al.*, 1993).

The prepared discrete binary system may be implemented with the use of digital systems. The ways of realisation of the interpreted Petri net with the application of the described encoding methods are presented in the next Subchapter.

2.2.2. The Realisation of a Petri Net in the Form of Digital Systems

The Subchapter presents alternative methods of the realisation of a concurrent automaton described with an interpreted Petri net in digital systems. A general idea as well as the most important aspects of particular solutions are discussed. A detailed description of the manner of the implementation of an interpreted Petri net is discussed in Subchapter 6.1.3.

2.2.2.1. The Implementation of a Concurrent Automaton with "One-Hot" Encoding

A concurrent automaton, in which the "one-hot" encoding method was used, is realised in digital systems as a single block. Each local state is represented by one flip-flop, value "1" refers to an active state, value "0" - inactive state. The transitions between the particular states are realised with the use of combinational logic, with the regard to binary values of the input signals. The state of outputs may be conditioned by the values of active inputs and local states (Mealy-type outputs (Mealy, 1955)), or may depend exclusively on the current values of local states (for Moore-type outputs (Moore, 1956)). The idea of a concurrent automaton with "one-hot" encoding is demonstrated in fig. 2.7.

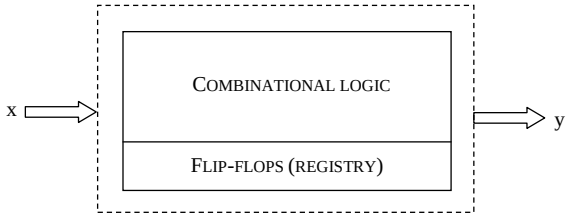

Fig. 2.7. Schematic diagram of a concurrent automaton with "one-hot" encoding

The main disadvantage of a "one-hot"-encoded automaton is the number of flip-flops which is closely related to the number of local states. Moreover, the concurrency of local places, in case of such encoding, definitely hinders the organisation and storage of outputs as structural memory, since it would be possible then to request the access to various memory cells (microinstructions) simultaneously (when more than one local state is active). Therefore, the frequently used solution involves parallel encoding (concurrent fragments of a net are encoded independently within a mutual encoding vector) or the realisation of a concurrent automaton in the form of a modular digital system (decomposition of a concurrent automaton into sequentially interlinked classical automata).

2.2.2.2. The Implementation of a Concurrent Automaton with Parallel Encoding

Modular parallel encoding of a concurrent digital automaton is based on a parallel decomposition of a discrete system. In the process, a concurrent automaton described by an interpreted Petri net is divided into independent concurrent subnets (state machine components). In the encoding process, each of the resultant subnets is treated independently, with the use of generally known encoding methods of classical sequential automata (Finite State Machine). For the places which occur in several subnets, *superposition* of codes occurring in subnets is performed (Banaszak *et al.*, 1993).

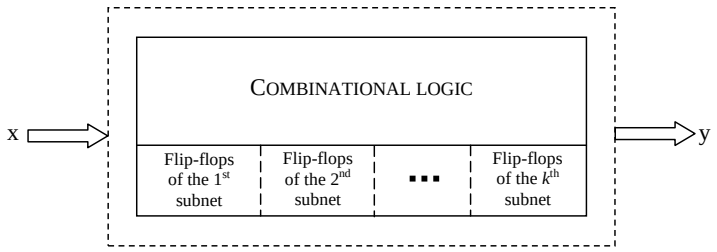

Fig. 2.8. Schematic diagram of a concurrent automaton with parallel encoding

The basic advantage of a concurrent automaton with parallel encoding is the reduction of the number of flip-flops necessary for the system implementation, when compared to "one-hot" encoding. The total number of flip-flops depends on the applied method of encoding of the internal states of particular subnets. A schematic diagram of a concurrent automaton with parallel encoding is presented in Fig. 2.8 (it has been assumed that the system consists of k-parallel subnets).

2.2.2.3. The Implementation of a Petri Net in the Form of a Modular Digital System

The last method of the implementation of a concurrent automaton presented in the book is the realisation of a system after a parallel decomposition, with the use of independent sequential automata. A parallel decomposition is described in details in Chapter 6, this chapter includes only a brief, general idea of the implementation of a prototyped system in digital devices.

The discussed method of the realisation of a concurrent automaton is based on parallel encoding. First, a concurrent automaton described by an interpreted Petri net is decomposed into SMCs. Next, each of the obtained subnets is treated as an independent sequential automaton, where it is indispensable to ensure the communication between particular automata in order to maintain the concurrency relation (Adamski, 1991; Banaszak *et al.*, 1993; Adamski and Barkalov, 2006). Moreover, an independent realisation of a system in the form of sequential automata results in the possibility to organise outputs of particular subnets in the

form of a memory (Wiśniewski, 2009). Thus, the concurrent fragments of a net are supplied with an independent memory and so any possible conflicts of the access to particular macrooperations are prevented. A schematic diagram of the discussed method of the implementation of a concurrent automaton in digital systems is presented in Fig. 2.9 (it has been assumed that the system was decomposed into k-parallel subnets).

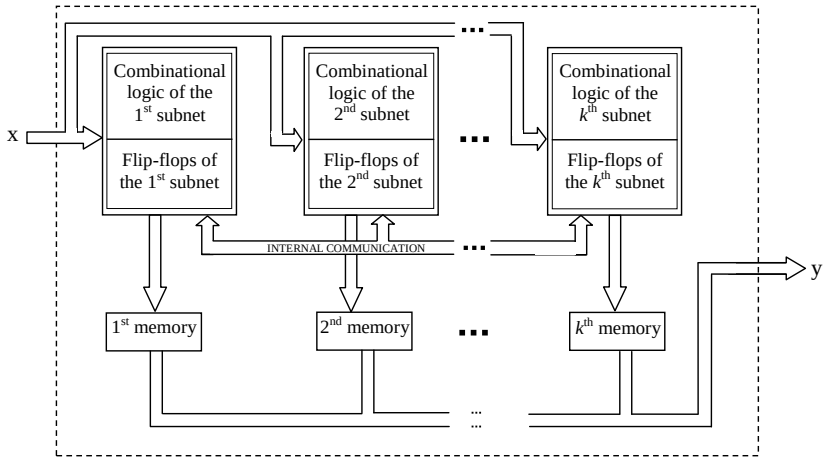

Fig. 2.9. Schematic diagram of a modular digital system

Chapter 3

UNDIRECTED GRAPHS AND HYPERGRAPHS

3.1. Graphs

The research on the theory of graphs dates back to the 16th century. Leonhard Euler, a Swiss mathematician and physicist who formulated the first theorem, is regarded to be the pioneer of the field. His inspiration came from the desire to find a path enabling a walk around Królewiec (Kaliningrad) in such a way that each bridge would be crossed just once. Euler proved that it was impossible. In his solution, the mathematician illustrated the problem with a graph, simultaneously formulating a general mathematical theorem connected with the graph cohesion. This example demonstrates the practicality of the theory of graphs, which are used today in numerous fields of science (Wilson, 1979; Harary, 1994).

Formally ***Graph G*** is defined by a pair:

$$G = (V, E), \tag{3.1}$$

where:
V=$\{v_1, \ldots, v_n\}$, is a finite, non-empty set of vertices;
E=$\{E_1, \ldots, E_m\}$, is a finite set of unordered pair of vertices, called edges (Berge, 1973). Exemplary graph G_1 is presented in Fig. 3.1.

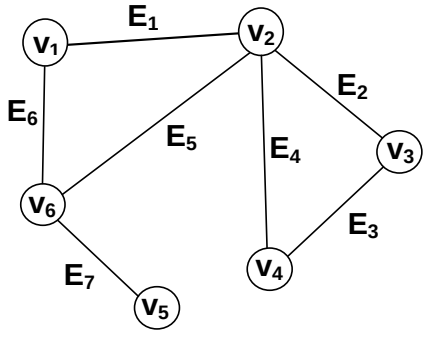

Fig. 3.1. Exemplary graph G_1

The presented graph consists of n=6 vertices: V=$\{v_1, \ldots, v_6\}$ and m=7 edges: E=$\{E_1, \ldots, E_7\}$. According to the presented definition, the edges form a set of unordered pairs of vertices. In the considered example: E_1=$\{v_1, v_2\}$, E_2=$\{v_2, v_3\}$, E_3=$\{v_3, v_4\}$, E_4=$\{v_2, v_4\}$, E_5=$\{v_2, v_6\}$, E_6=$\{v_1, v_6\}$ and E_7=$\{v_5, v_6\}$.

It should be clearly stated here, that any descriptions or definitions connected with the *graphs* mentioned in this book refer to *undirected graphs*, in which the set of edges consists of *unordered pairs of vertices*, as opposed to directed graphs, whose edges are composed of *ordered pairs of vertices*.

One of the methods of graph representation is *an incidence matrix*, with rows referring to edges, and columns referring to graph vertices. If a matrix element equals 1, the i-th edge ($i \in 1, \ldots, m$) is incident to the j-th vertex ($j \in 1, \ldots, n$). Otherwise, the element equals 0:

$$A = \begin{cases} 1 & \text{if} \quad v_j \in E_i \\ 0 & \text{if} \quad v_j \notin E_i \end{cases}, \tag{3.2}$$

Incidence matrix A_1 for graph G_1 is shown in Fig. 3.2. Since graph G_1 comprises 7 edges, matrix A_1 has 7 rows. Analogically, six vertices are represented by six columns of matrix A_1.

$$A_1 = \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \end{array}$$

Fig. 3.2. Incidence matrix of graph G_1

Another widely used representation of graphs is *a neighborhood matrix*. In this case, it is a symmetric matrix including relations between particular vertices. The entry in the i-th row and j-th column of the matrix determines the number of edges joining the i-th and the j-th vertices.

Neighborhood matrix N_1 for graph G_1 is presented in Fig. 3.3.
The most important definitions connected with the graph theory which are used in the work are presented below (Berge, 1973; Korzan, 1978; Wilson, 1979; Harary, 1994).

Definition 3.1. ***Subgraph*** *P* ***of graph*** *G is a graph formed by removing vertices or edges from graph G.*

For example, subgraph P_1 consisting of n=6 vertices: V=$\{v_1, \ldots, v_6\}$ and m=6 edges: E=$\{E_1, E_2, E_3, E_4, E_6, E_7\}$ may be obtained by removing edge E_5

$$N_1 = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix} \\ \left[\begin{array}{cccccc} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{matrix}$$

Fig. 3.3. Neighborhood matrix of graph G_1

from graph G_1. Other examples may be: subgraph P_2, comprising three vertices $\{v_1, v_2, v_6\}$ and three edges $\{E_1, E_5, E_6\}$, as well as subgraph P_3 consisting of the same vertices as P_2 but including only two edges E_1 and E_6.

Definition 3.2. ***Complete graph*** *K_n is a graph where each of n vertices is connected by an edge with all the remaining vertices. The number of edges in a complete graph equals:*

$$m_{K_n} = \frac{n * (n-1)}{2}. \tag{3.3}$$

An example of a complete graph is subgraph P_2, presented above (see Definition 3.1). In this case all three vertices are joined with each other by edges. According to the definition, the total number of edges in such a subgraph equals m_{K_3}=$3 * (3-1)/2$=3. Indeed, subgraph P_2 consists of three edges: $\{E_1, E_5, E_6\}$.

Definition 3.3. ***Clique*** *Cl is a subnet of vertices of graph G, where each two vertices are connected with an edge. In other words, clique Cl is a complete subgraph of graph G.*

The determination whether there is a clique in a graph of a given size, is an NP-complete problem. There is no algorithm enabling the determination of all the cliques in a graph in a polynomial time (Berge, 1973; Harary, 1994).

An example of clique Cl_1 in graph G_1 is subgraph P_2. Additionally, graph G_1 contains one more clique consisting of three vertices: Cl_2=$\{v_2, v_3, v_4\}$.

Definition 3.4. ***The complement of graph*** *$G = (V_G, E_G)$ is graph $\overline{G} = (V_G, E_L)$, of the same set of vertices V_G, and the same set of edges E_L, which is the complement of set E_G. It means that there is an edge between vertices of graph $\overline{G}$ if and only if there is no edge between these vertices in graph G.*

The complement of graph G_1, presented in Fig. 3.4, is graph $\overline{G_1}$, which, just as graph G_1 contains six vertices: V=$\{v_1, \ldots, v_6\}$. The edges in graph $\overline{G_1}$ occur only when there are no connections between vertices of graph G_1, and so the set of edges in $\overline{G_1}$ consists of eight elements: E_L=$\{E_8, \ldots, E_{15}\}$. It is worth mentioning that the total number of edges in graphs G_1 and $\overline{G_1}$ is equal to the number of edges of a complete graph of V=6 vertices, that is m_{K_6}=15.

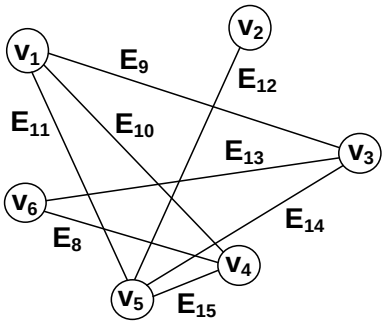

Fig. 3.4. Graph $\overline{G_1}$ (the complement of graph G_1)

Definition 3.5. ***Vertex cover of graph*** *$G = (V, E)$ is set $V' \subseteq V$ containing vertices incidental to each graph edge. The smallest subset of vertices of a graph incident to all the graph edges is called the smallest vertex cover.*

Exemplary graph vertex cover is shown in Fig. 3.5 (a).

Definition 3.6. ***Edge cover of graph*** *G is such a subset $E' \subseteq E$ of its edges in which each vertex of graph G is incident to at least one edge from this subset. The smallest subset of graph edges incident to all its vertices is called the smallest edge cover.*

An exemplary graph cover is shown in Fig. 3.5 (b).

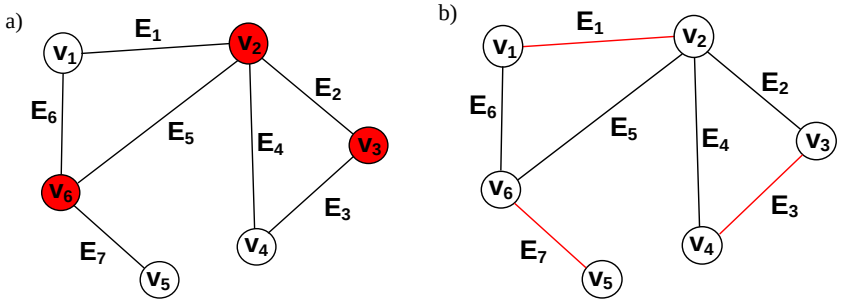

Fig. 3.5. Exemplary graph cover *a*) vertex cover *b*) edge cover

Definition 3.7. ***Graph coloring*** *or* ***graph vertex coloring*** *is the assignment of one selected color to each graph vertex such that no two adjacent vertices (i.e., such vertices which are joined by an edge) share the same color.*

Definition 3.8. ***Chromatic number*** $\chi(G)$ *is equal to the smallest number of colors which the graph can be colored with (i.e., the smallest possible k-coloring of a graph).*

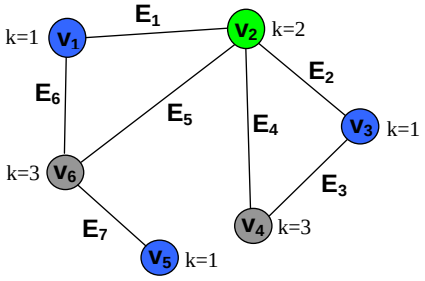

Fig. 3.6. Coloring of G_1

Figure 3.6 demonstrates one of the possible ways of coloring of G_1. In the presented example the smallest possible number of colors the graph can be colored with equals $\chi(G_1)$=3. The first color was used for vertices v_1, v_3 and v_5, the second color was used for v_2, whereas the third color for v_4 i v_6.

The determination of a chromatic number (i.e., finding a solution with the smallest number of colors) is an NP-complete problem. As in the case of graph covering, there are numerous methods of graph coloring (exact, stochastic etc.). The basic algorithms of graph coloring are described in Chapter 4.

Definition 3.9. ***A color class*** *is a set of all the graph vertices marked with the same i-th color.*

Definition 3.10. ***A compatibility class*** C_i ***(an independent set)*** *is a set of graph vertices, which are not connected by any edge. Compatibility class* C_i *is called* ***a maximum*** *(or* ***a maximal independent set****) if there is no class* C_j *such that* $C_j \supset C_i$.

Unlike the color classes, various compatibility classes may contain the same graph vertices. Thus it means that a color class constitutes simultaneously a set (or a subset) of a graph compatibility class, however, the reverse dependency need not be true.

Definition 3.11. ***A set of compatibility classes*** C_C *is a set of all compatibility classes of graph* $C_C=\{C_1,\ldots,C_K\}$.

For the coloring of G_1 shown above there exist K=5 independent sets $C_C=\{C_1,\ldots,C_5\}$. Particular classes include the following vertices: $C_1=\{v_1,v_3, v_5\}$, $C_2=\{v_1,v_4,v_5\}$, $C_3=\{v_2,v_5\}$, $C_4=\{v_3,v_6\}$, $C_5=\{v_4,v_6\}$. Therefore, classes C_1 and C_5 refer to the first and third color classes of the graph, whereas the only vertex marked with the second color is simultaneously a subset of the third compatibility class.

3.2. Hypergraphs

The notion of a hypergraph was first used in the second half of the previous century. In 1973, a French mathematician, Claude Berge, published a monograph "Graphs and Hypergraphs" in which he formalised and uniformed the basic definitions of the theory of hypergraphs.

From the formal point of view, a hypergraph is an extension of the idea of a graph. Its edges, called hyperedges, may be incident to an arbitrary number of vertices (Berge, 1973).

Hypergraph H is defined by a pair:

$$H = (V, E), \tag{3.4}$$

where:
V=$\{v_1, \ldots, v_n\}$, is an arbitrary, non-empty set of vertices;
E=$\{E_1, \ldots, E_m\}$, is a set of hypergraph edges, i.e., a subset of set $P(V)$ of all the possible non-empty sets, elements of which belong to V.

An example of hypergraph H_1 is presented in Fig. 3.7.

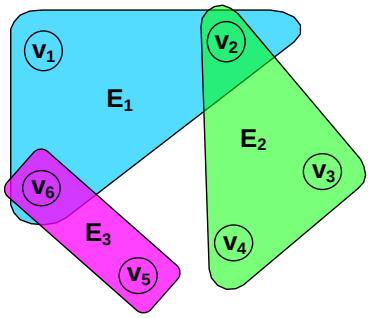

Fig. 3.7. Exemplary hypergraph H_1

There are m=3 hyperedges: E=$\{E_1, \ldots, E_3\}$ in hypergraph H_1. According to the presented definition of a hypergraph, each hyperedge is a non-empty set of vertices. For H_1 edge E_3 contains two vertices, whereas hyperedges E_1 and E_2 are incident to three vertices. This is the basic difference in comparison to graphs, edges of which may be incident to maximum two edges. The presented hypergraph is a generalisation of graph G_1. Edges E_1, E_5 and E_6 form a clique in graph G_1, and in the hypergraph they are represented by single hyperedge E_1. Analogically, edges E_2, E_3 and E_4 refer to hyperedge E_2.

In *the incidence matrix A* of a hypergraph, the rows refer to hypergraph edges and the columns refer to its vertices. If a matrix element equals 1, i-th edge ($i \in 1, \ldots, m$) is incident to j-th vertex ($j \in 1, \ldots, n$). Otherwise the element equals 0:

$$A = \begin{cases} 1 & \text{if} \quad v_j \in E_i \\ 0 & \text{if} \quad v_j \notin E_i \end{cases}, \tag{3.5}$$

Incidence matrix A_1 for hypergraph H_1 is shown in Fig. 3.8. Since hypergraph H_1 consists of three edges, matrix A_1 has three rows. Analogically, six vertices are represented by the columns of matrix A_1.

$$A_1 = \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} E_1 \\ E_2 \\ E_3 \end{array}$$

Fig. 3.8. Incidence matrix for hypergraph H_1

Definition 3.12. *A hypergraph in which no edge contains any other edges is called* ***a simple hypergraph****. From a formal point of view, hypergraph $H = (V, E)$ is a simple hypergraph if for an arbitrary hyperedge $E_i \in E$, there is no hyperedge $E_j \in E$ such that $E_i \subset E_j$.*

Definition 3.13. ***The complement of hypergraph*** *$H = (V, E)$ is hypergraph $\overline{H} = (V, \overline{E})$ of the same set of vertices as the hypergraph H and the set of hyperedges $\overline{E}$, being the complement of set E. Vertices of hypergraph $\overline{H}$ are connected by a hyperedge if and only if the vertices are not connected by a hyperedge in hypergraph H.*

Figure 3.9 presents a complement of hypergraph H_1 shown in Fig. 3.7. Hypergraph $\overline{H_1}$ consists of $\overline{E}$=5 edges, two of which (E_4 oraz E_5) are incident to three vertices, whereas the remaining three contain two vertices each.

Definition 3.14. *For each hypergraph $H = (V, E)$, there is* ***dual hypergraph*** *H^*=(E, V), edges of which correspond to vertices of hypergraph H, whereas vertices refer to its edges. The incidence matrix A^* of the dual hypergraph H^* is a transposed matrix A of hypergraph H. Analogically, matrix A of hypergraph H is a transposed matrix A^* of hypergraph H^*.*

Definition 3.15. ***A transversal (hitting set, vertex cover) of hypergraph*** *H is set $T \subseteq V$ containing vertices incident to each edge of the hypergraph (Berge, 1989; Eiter and Gottlob, 2002).* ***A minimal transversal*** *is such a transversal which contains no other transversal of hypergraph H (Eiter and Gottlob, 1995).*

Definition 3.16. ***The smallest transversal*** *$\tau(H)$ of hypergraph H is a transversal with the smallest number of vertices of all the transversals of hypergraph H.*

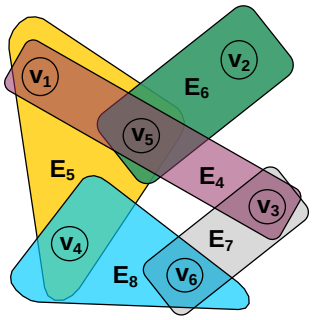

Fig. 3.9. Complement of hypergraph H_1

Definition 3.17. ***The exact transversal*** *D* ***of hypergraph*** *H is set $D \subseteq V$ of vertices of hypergraph H, which is incident to all edges of hypergraph H, where each edge is incident to exactly one vertex of set D.*

Definition 3.18. ***The smallest exact transversal*** *δ of hypergraph H is an exact transversal containing the smallest number of elements of all the exact transversals of hypergraph H.*

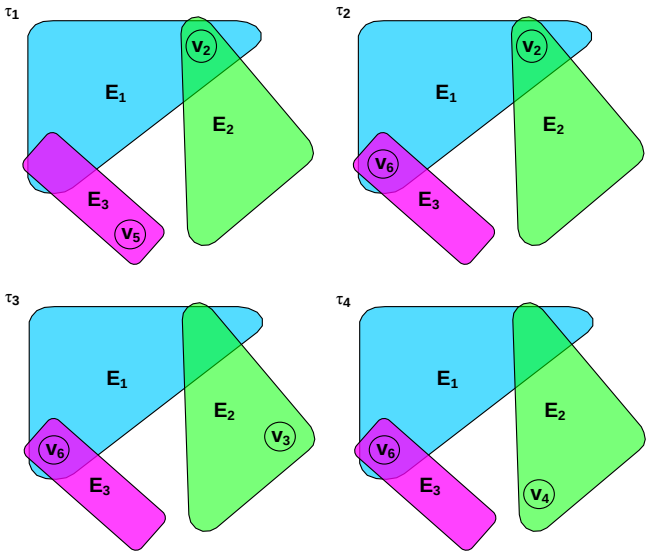

Fig. 3.10. Transversals of hypergraph H_1

Fig. 3.10 presents all possible smallest transversals of hypergraph H_1. In the presented example, there are four smallest transversals: $\tau_1=\{v_2, v_5\}$, $\tau_2=\{v_2, v_6\}$,

τ_3=$\{v_3, v_6\}$ and τ_4=$\{v_4, v_6\}$. It is worth adding that three out of the above coverings are simultaneously the smallest exact transversals (τ_1, τ_3 and τ_4), since each of the remaining vertices is incident to one edge. In the case of τ_2, vertices v_2 and v_6 are incident to the same edge E_1, so that they do not form an exact transversal. More information connected with transversals and basic methods of their determination may be found in Chapter 4.

There is no direct relation between the vertex cover of graphs and transversals of hypergraphs. The above considerations imply that the smallest cover of graph G_1 is realised by three vertices, whereas the smallest transversal is a two-element set in the corresponding hypergraph H_1. The difference results from the hypergraph structure, in which hyperedge E_2 includes both vertex v_2, and v_4. More information on the relation between graphs and hypergraphs may be found in Subchapter 4.3.

Definition 3.19. ***An essential vertex*** *- is the only vertex that belongs to a given edge. Essential vertices must be a part of each transversal (Rudell, 1989; DeMicheli, 1994).*

Definition 3.20. ***An exact transversal hypergraph*** *(xt-hypergraph) is a hypergraph in which all minimal transversals are also exact transversals.*

The determination of the first and the subsequent exact transversals in a hypergraph is connected with the exponential calculation complexity. It has been proved (Eiter, 1994; Elbassioni and Rauf, 2010) that the determination of the first and the subsequent transversals in an exact transversal hypergraph may be performed in polynomial time. Moreover, it has been revealed that determining whether a given hypergraph is an exact transversal hypergraph may also be performed in polynomial time.

All the problems connected with the calculation complexity of algorithms discussed in the book are interpreted as time complexity of an algorithm, which is determined as a relation between the size of input data and the number of operations performed by the algorithm.

Definition 3.21. ***A c-exact hypergraph*** *is a hypergraph in which a pair of compatible vertices (not connected with an edge) belongs to at least one exact transversal.*

A c-exact hypergraph is a generalisation of an exact transversal hypergraph. In this case, the condition that each minimal transversal is also an exact transversal does not have to be satisfied. However, any two arbitrary vertices which are in a compatibility relation (i.e., not connected by an edge) must belong to at least one exact transversal. Hence, each vertex of the c-exact hypergraph must belong to at least one exact transversal.

The notion of a c-exact hypergraph has not occurred in the literature so far and has been introduced for the needs of the book. It is shown in Chapter 5 that the determination of the first and the subsequent transversals for a c-exact hypergraph is connected with a polynomial calculation complexity, as in the case of the exact transversal hypergraph.

Hypergraph H_1, presented in Fig. 3.7, is a c-exact hypergraph. Each pair of vertices which are compatible is included in an exact transversal. For example, vertices v_1 and v_3 are compatible. They do not form a transversal (edge E_3 not covered) thus, there must be a vertex which is exclusively incident to this particular edge. In fact, vertex v_5 is complementary to elements v_1 and v_3, which together form an exact transversal. Hypergraph H_1 on the other hand, is not an exact transversal hypergraph where all minimal transversals are also the exact ones. The transversal formed by vertices v_2 and v_6 is minimal, but not exact.

A c-exact hypergraph has very interesting properties. The removal of any arbitrary vertex (along with incidence edges and other vertices belonging to these edges) results in the reduced hypergraph which is also c-exact. Moreover, the complement to a c-exact hypergraph also constitutes a c-exact hypergraph. This fact is of a great importance for the relations describing two complementary relations, such as compatibility - incompatibility, concurrency - sequentiality, etc. The exemplary hypergraph $\overline{H_1}$, presented in Fig. 3.9, is a c-exact hypergraph since it constitutes a complement to hypergraph H_1.

The detailed analysis of the above-mentioned relations and the sketches of the theorems and proofs are presented in Chapter 5.

Definition 3.22. ***An edge cover of hypergraph*** *H is a set of hyperedges incident to all vertices of the hypergraph.*

In the theory of hypergraphs, an edge cover used as a mathematical apparatus is definitely less frequent than transversals. It results from the fact that an edge cover of hypergraph H may be replaced with a notion of a vertex base of a dual hypergraph H^*. Therefore in practice, the research connected with hypergraph covering is reduced to finding hypergraph transversals (or dual hypergraph transversals, if there is a need).

Definition 3.23. ***Vertex coloring of a hypergraph****, also referred to as* ***strong hypergraph coloring*** *or merely* ***hypergraph coloring****, is the assignment of a selected color to each hypergraph in such a way that two adjacent vertices (i.e., such vertices which are joined by a hyperedge) are of a different color.*

There is a close relation between exact transversals and hypergraph coloring. For c-exact hypergraphs, coloring may be carried out with the help of exact transversals. Then, vertices combined in the same exact transversal are marked with the same color (if a given vertex occurs in several transversals, it is removed from other exact transversals). This problem is not widely discussed in the book, however, it may serve as a base for the further research (taking into account the polynomial character of the process of determining subsequent exact transversals).

Definition 3.24. ***Edge coloring of a hypergraph****, also referred to as* ***weak coloring of a hypergraph****, is the assignment of a selected color to each hypergraph edge in such a way that two neighboring edges (i.e., the ones that contain common vertices) are of a different color (Kubale et al., 2006).*

The works and research presented in the book focus on strong coloring of hypergraphs. Therefore all terms and issues connected with hypergraph coloring refer to **strong hypergraph coloring**.

Definition 3.25. ***A strong chromatic number*** *or merely* ***chromatic number*** $\chi(H)$ *is the value equal to the smallest number of colors the hypergraph vertices may be colored with.*

Definition 3.26. ***A color class*** *is a set of all the vertices of a hypergraph marked with the same color.*

Definition 3.27. ***A compatibility class*** C_i ***(an independent set)*** *is a set of these vertices of a hypergraph which are not connected by a hyperedge. A compatibility class* C_i *is called* ***a maximum compatibility class*** *(or* ***a maximal independent set****), if there is no class* C_j *such that* $C_j \supset C_i$ *(Eiter, 1994).*

Definition 3.28. ***A set of compatibility classes*** C_C *is a set of all compatibility classes of hypergraph* $C_C=\{C_1,\ldots,C_K\}$.

It should be clearly stated that definitions 3.9 and 3.26 as well as 3.10 and 3.27 are interrelated. In practice, the set of color classes of a graph is equal to the set of color classes in a corresponding hypergraph. Analogically, the set of independent classes in a graph corresponds to the set of independent classes in a hypergraph. A detailed analysis of the dependencies between graph and hypergraph algorithms is described in Subchapter 4.3.

Fig. 3.11 presents an exemplary coloring of hypergraph H_1. The smallest possible number of colors has been used, the chromatic number equals $\chi(H_1)$=3. There is a close connection between graph and hypergraph coloring. The presented results of the coloring of hypergraph H_1 are identical to the results obtained for the coloring of graph G_1. It results from the fact that there is a relationship between coloring and cliques occurring in a graph. Vertices comprised in a given clique cannot be colored with the same color, therefore naturally the same relationship occurs for hypergraph edges. A detailed comparison of graph and hypergraph coloring is presented in Chapter 4.

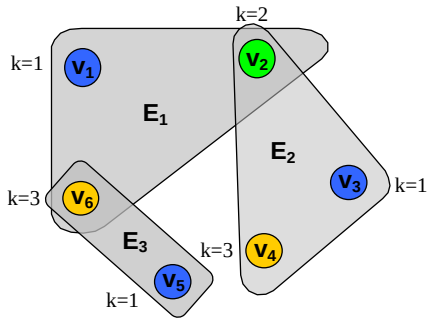

Fig. 3.11. Example of hypergraph H_1 coloring

Chapter 4

GRAPH AND HYPERGRAPH ALGORITHMS

The chapter presents mathematical methods of examining graph and hypergraph structures including the determination of complements, coloring or covering. The operations are the key elements of the decomposition methods for discrete systems, developed in the book.

All the presented methods have been implemented with the use of C++ (Kernighan and Ritchie, 1977; Stroustrup, 1986) **and have been combined in the *Hippo* system** (the program is presented in Appendix A). There are numerous methods of solving the same problem, e.g., various algorithms of covering, coloring etc. (Cormen *et al.*, 2001), therefore the work includes only these methods which were realised and implemented in the *Hippo* system.

4.1. Mathematical Methods of Examining a Graph

4.1.1. A Graph Complement

According to Def. 3.4, a graph complement is a graph containing the same vertices, but its edges occur between vertices which have not been connected in the primary graph (Berge, 1973; Harary, 1994). It implies that one of the simplest methods of the complement determination may be the application of the graph adjacency matrix. Unlike the incidence matrix, it stores all information about the connections between particular vertices, hence, in practice, finding the complement is reduced to replacing the content of the matrix.

Formally, an algorithm to determine the complement of graph G may be described as follows:

1. The presentation of graph G with the use of adjacency matrix N.
2. The replacement of the values of all the elements in matrix N from 1 to 0 and from 0 to 1, for elements lying above the diagonal of matrix N (elements lying below the diagonal are filled symmetrically).

The obtained adjacency matrix $\overline{N}$ represents the values of graph $\overline{G}$, which is the complement of graph G. The graphical representation of the graph complement is very intuitive. The formed graph $\overline{G}$ contains connections (edges) between these vertices which are not adjacent in the initial graph G.

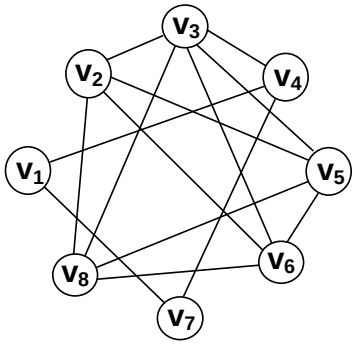

Fig. 4.1. Graph G_2

A complement of graph G_2 shown in Fig. 4.1, will be determined as an example. The presented graph consists of $|V|$=8 vertices and $|E|$=14 edges. In order to improve legibility, the names of the edges have been ignored in the graphical presentation, leaving merely the labels of particular vertices.

According to the presented algorithm, the first step is to determine the adjacency matrix. In the discussed example, it has eight rows and eight columns. The adjacency matrix N_2 of graph G_2 is illustrated in Fig. 4.2.

$$N_2 = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{matrix}$$

Fig. 4.2. Adjacency matrix of graph G_2

Next, the complement of graph G_2 is determined. The particular elements of the adjacency matrix are modified. If a given element lies above the main diagonal of the matrix, its value is altered (ones are replaced with zeros, whereas - zeros with ones). Since the matrix is symmetric, the elements below the main diagonal are modified simultaneously. The final form of adjacency matrix $\overline{N_2}$ for the graph being the complement of graph G_2 is presented in Fig. 4.3.

$$\overline{N_2} = \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\ \left[\begin{array}{cccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{array}$$

Fig. 4.3. Adjacency matrix $\overline{N_2}$ of graph $\overline{G_2}$

It results from matrix $\overline{N_2}$ that graph $\overline{G_2}$ contains $|V|$=8 vertices and $|\overline{E}|$=14 edges. The total number of edges in graphs G_2 and $\overline{G_2}$ should be equal to the number of edges in a complete graph with eight vertices. The relation is satisfied (m_{K_8}=$|E_{G_2}|$+$|\overline{E_{G_2}}|$=28), which means that the complement has been determined properly. The graphical illustration of the complement $\overline{G_2}$ is presented in Fig. 4.4.

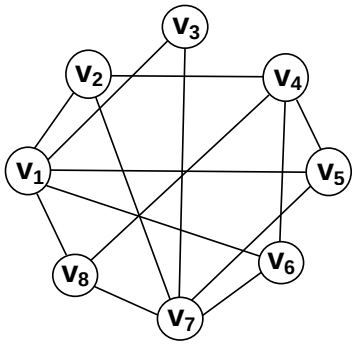

Fig. 4.4. Complement of graph G_2

4.1.2. A Graph Vertex Cover

The problem of a vertex cover is reduced to finding a subset of vertices in such a manner that each edge is incident to at least one vertex from this subset. The smallest vertex cover is the smallest possible set of vertices realising the graph cover. Finding the smallest vertex cover in a graph is classified as an NP-complete problem (Berge, 1973). It means that the exact solution, allowing the smallest vertex cover to be found in the polynomial time, is currently unknown. For small graphs, the cover problem may be solved with the use of the backtracking algo-

rithm. The method checks all possible combinations, therefore it gives the best results, i.e., the smallest vertex cover in the graph. An essential disadvantage of the backtracking algorithm is its computational complexity which is exponential. It means that in practice the method is limited to relatively small graphs. In this case, the application of other solutions seems to be helpful. One of the most efficient solution is the greedy algorithm, which finds a global result through a local-decision-making process. Its greatest advantage is its computational complexity (the method operates in a quasi-linear time) (Aho *et al.*, 1974; Papadimitriou, 1994; Knuth, 1997). Unfortunately, the found solution is not always the best one. What is more, it may happen that the algorithm gets stuck in a local minimum (DeMicheli, 1994). The book presents these two methods of the determination of a graph vertex cover in detail.

4.1.2.1. A Backtracking Algorithm

A backtracking algorithm searches the possible set of solutions (DeMicheli, 1994; Sapiecha, 1999; Sysło *et al.*, 1983). A vertex marked with the lowest value (according to the lexicographic order) is selected and reduced from the set of graph vertices. If the remaining set of vertices still remains the graph cover, and the obtained result is better than the one found so far, the current cover becomes the best one. In the subsequent stages, the algorithm recursively searches for the best solution for all the vertices.

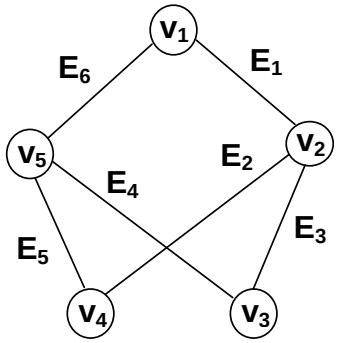

Fig. 4.5. Graph G_3

For graph G_3, illustrated in Fig. 4.5, the set of vertices consists of five elements $V=\{v_1,\ldots,v_5\}$. According to the presented algorithm, vertex v_1 is selected for the analysis as the first one. After the reduction, the graph consists of four vertices: $V^1=\{v_2,\ldots,v_5\}$. Since the remaining elements still form a cover, the set becomes the best currently found solution, thus $V'=V^1=\{v_2,\ldots,v_5\}$. Next, vertex v_2 is removed, thus the vertex set is reduced to $V^2=\{v_3,\ldots,v_5\}$. The resulting set does not form a cover since edge E_1 is not incident to any vertex comprised in V^2. Therefore, the algorithm comes back to the previous solution (V^1) iand continues

searching for the cover, reducing the subsequent vertex, i.e., v_3. The remaining graph vertices form set V^3=$\{v_2, v_4, v_5\}$ incident to all the edges. The solution is better than the previous one, thus set V^3 becomes the best found solution. In the subsequent phase, vertex v_4 is reduced and only vertices V^5=$\{v_2, v_5\}$ remain in the graph. Thus this set is a cover which is better than the previously found ones. In the further stage, vertex v_5 is eliminated, therefore V^6=$\{v_2\}$. The set does not compose a cover, so solution V^5 remains the best cover.

The algorithm recursively comes back to the previous states, checking all possible solutions. The subsequent stages are performed analogically to the ones presented above. Eventually, set V^5 appears to be the best cover. The set contains two vertices (v_2 and v_5), which are incident to all six edges of graph G_3 (Fig. 4.6).

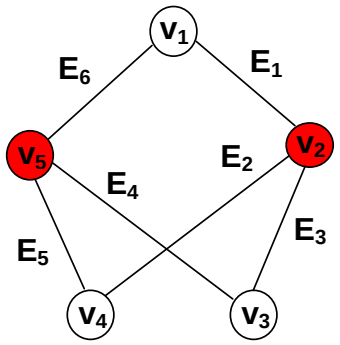

Fig. 4.6. Vertex cover of graph G_3

4.1.2.2. A Greedy Algorithm

The algorithm presented in the previous chapter always returns the best solution. An essential drawback of the method is its exponential computational complexity, which frequently means that the given problem may not be solved or the solution may be obtained in an unsatisfactory time.

A greedy algorithm searches for a solution on the basis of the locally optimal decision (DeMicheli, 1994; Sapiecha, 1999; Hoos and Stützle, 2000). It means that in each step, the current best possibility is selected, and it does not matter if the decision is globally optimal in a given moment (hence the name ″greedy"). The indisputable advantage of the algorithm is its computational complexity. However, the method does not always find the best solution, i.e., the smallest vertex cover of a graph.

The rule for a greedy algorithm is as follows:

1. *Determine the value of the covering set* V'=$\emptyset$. At the beginning the searched set of the cover is empt, the vertices will be determined and added in the subsequent cycles.

2. *Select an essential vertex, if such does not exist - select a vertex with the largest degree.* At this stage, the algorithm carries out a local selection of the optimal solution. The essential vertex is selected (which has to be a part of the searched cover). If there is no such vertex, a vertex incident to the largest number of edges is selected for the further analysis. If there are more than one solution (e.g., there are two essential vertices or two 0), then the algorithm realises the selection according to the lexicographic order.

3. *Add the selected vertex to the cover set V', and then remove the vertex from the graph together with all edges incident to it.*

4. *Check if V' forms a cover, if not - repeat the algorithm from step 2.* In this step, set V' is checked. If the vertices comprised in the set form the searched vertex cover, the algorithm finishes its operation. Otherwise, the method undergoes another cycle in which the next local optimum is searched (i.e., the subsequent essential vertex or a vertex with the largest degree).

According to the presented scheme, searching for the cover with the greedy method always starts with the determination of the value of set $V'=\emptyset$. Next, the local minimum is searched through the selection of a vertex with the largest degree (in the given graph, there exists no essential vertex). In the case of graph G_3 there are two vertices with the largest degree: v_2 and v_5. According to the lexicographic order, element v_2 is added to set V' and, simultaneously, removed from the graph together with the edges incident to it (i.e., E_1,E_3 and E_4). Thus, the graph has been reduced to four vertices and three edges (Fig. 4.7).

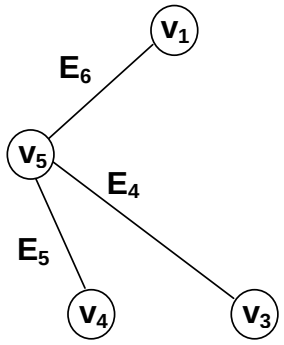

Fig. 4.7. Graph G_3 after the first stage of the greedy algorithm

In the subsequent step, it is checked if set V' is the cover of graph G_3. Since the relation is not satisfied, the cycle is repeated. Again, there exists no essential vertex, so a vertex with the largest degree is selected. Among the remaining vertices, there exists only one vertex v_5 incident to three edges, whereas the remaining vertices are of the first degree. It means that v_5 is added to V', and then it is removed from the graph. Since set V' realises the vertex cover, the algorithm is

finished. The obtained solution $V'=\{v_2, v_5\}$ is simultaneously the smallest cover, and the same result has been obtained for the backtracking algorithm.

$$A_Z = \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \end{array}$$

Fig. 4.8. Incidence matrix A_Z

The example presented below shows that although the greedy algorithm finds a cover, it is not the smallest one. The analysis includes an exemplary graph with incidence matrix A_Z presented in Fig. 4.8. The graph has $|V|$=7 vertices and $|E|$=6 edges. There exists a smallest vertex cover. The set consists of three vertices: $V'=\{v_2, v_3, v_4\}$. Such a solution may be obtained by applying the backtracking algorithm. However, in the case of the greedy algorithm, the result is different, which is demonstrated below.

In the graph there exists no essential vertex, so in the first cycle of the greedy method, a vertex with the largest degree is selected. In this case, it is vertex v_1, which is incident to three edges: E_1, E_3 and E_5. The vertex is added to the searched set V', whereas the graph is reduced. Both vertex v_1, and all the edges incident to it are removed. As a result of the reduction, there occurs a new structure with an incidence matrix presented in Fig. 4.9.

$$A'_Z = \begin{array}{c} \begin{array}{cccccc} v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ E_2 \\ E_4 \\ E_6 \end{array}$$

Fig. 4.9. Incidence matrix A_Z after the removal of vertex v_1

Since the initial graph cover has not been found, the greedy algorithm repeats the procedure of vertex selection. Again, there exists no essential vertex, so the vertex of the largest degree is taken into account for further consideration. It results from Fig. 4.9 that all the remaining vertices are first degree vertices, so the next vertex to be reduced will be the one resulting from the lexicographic order, i.e., v_2. Continuing the procedure, the algorithm reduces vertex v_3 in the next step and v_4 - in the last.

As can be seen from the above considerations, the greedy algorithm returns a set consisting of four vertices. The obtained result forms a graph cover which, however, is not a minimal one. It results directly from the method itself in which the decisions are exclusively made on the basis of local information. This is so in the case of the selection of vertex v_1, which indeed is incident to the largest number of edges, although its selection prevented the global minimum to be obtained, which, in the presented example, is a vertex cover consisting of three elements $V'=\{v_2, v_3, v_4\}$.

4.1.3. Graph Coloring

Graph coloring relies on marking all the graph vertices in such a way that the vertices connected with an edge were not of the same color. This chapter includes the presentation and comparison of most popular coloring algorithms (Kubale, 2002). The first four methods are based on the sequential vertex coloring and they search for an approximate solution. The advantage of such an approach is the computational complexity (the coloring process is reduced to a greedy determination of subsequent vertices). Unfortunately, sequential coloring does not always returns the best solution. The process of searching for the local optimum is improved by vertex ordering algorithms (LF, SL, sometimes random ordering returns very good results). By far the most effective graph coloring method is an exact method checking all the possible solutions (backtracking algorithms). The algorithm always returns the best solution (in practice, the graph is colored with the use of the smallest possible number of color classes), its basic disadvantage is its computational complexity of the exponential order.

4.1.3.1. Unordered Sequential Coloring

The unordered sequential coloring method browses the graph vertices one by one assigning the smallest possible color to each vertex. Initially, the number of colors k equals to zero. In the subsequent stages, as the vertices are marked, successive labels are added.

The sequential coloring procedure is as follows:

1. *Reset the colors of all the vertices and set the number of colors $k=0$.* Initially, the colors of all vertices are reset. What is more, variable k, storing the number of currently used colors is reset.

2. *Select the subsequent uncolored vertex v_i out of set V.* An uncolored vertex is selected (according to the lexicographic order) for the analysis needs.

3. *Assign the smallest possible color to vertex v_i.* If possible, a color with the smallest index from the set of colors that have been already assigned to other vertices is assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.

4. *Repeat steps 2 and 3 until all graph vertices are colored.* Finally, all graph vertices are marked with the use of k colors.

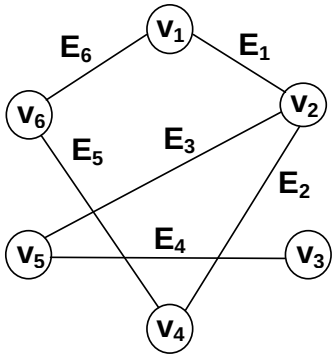

Fig. 4.10. Graph G_4

The presented algorithm has been illustrated by an example. Graph G_4 shown in Fig. 4.10 consists of $|V|$=6 vertices and $|E|$=6 edges. Initially k=0 and the colors of all vertices are reset. According to the presented algorithm, vertex v_1 is colored first. Since the previous set of colors is empty, the vertex is marked with color 1. The same value takes variable k, which stores the number of all used colors. In the next stage, vertex v_2 is selected for the analysis. The element is connected with edge v_1, therefore it cannot be marked with the first color. Consequently, value k is increased, and v_2 takes the second color.

The next phase is to mark vertex v_3. Since the element is not connected with v_1 by an edge, it is colored with the smallest possible color, i.e., the first one, and the general number of colors remains unchanged (k=2). A similar situation occurs for vertex v_4, which is colored with the same label as v_1 and v_3, since it is not adjacent to these elements. Vertex v_5 is a neighbor of both v_2 and v_3, therefore it cannot be marked with any of the previously used colors. Thus, the general number of colors is increased, and v_5 is marked with the third color. The last vertex v_6 is connected with v_1 and v_4, and cannot be colored with the first label. However, it is not a neighbor of v_2, so it is marked with the second color. Finally, graph G_4 is divided into k=3 sets (color classes). Vertices v_1, v_3 and v_4 are colored with the first color, vertices v_2 and v_6, with the second color, whereas the third color is used for vertex v_5 (Fig. 4.11).

4.1.3.2. Random Ordered Coloring

Random ordered coloring of graph vertices is a process analogical to the sequential coloring method. However, before performing the coloring process, the vertices are randomly selected. The main idea of random selection of vertices is the possibility to avoid local optima which frequently lead the algorithm to results which are not

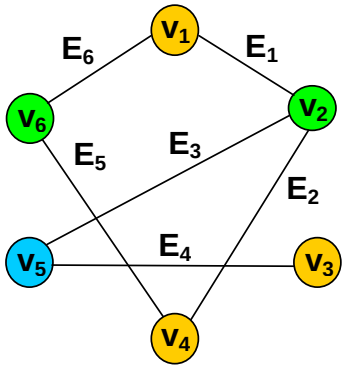

Fig. 4.11. Unordered coloring of graph G_4

globally best, but simultaneously allow maintaining the computational complexity at the same level as for the unordered sequential coloring.

Random ordered coloring may be divided into the following stages:

1. *Order the vertices randomly.* All the elements from set V are sorted in the random order.
2. *Reset the colors of all the vertices and set the initial number of colors k=0.*
3. *Select the subsequent uncolored vertex v_i from set V according to the order established previously.* An uncolored vertex is chosen for the analysis (according to the order established in point 1).
4. *Assign the smallest possible color to vertex v_i.* If possible, the color with the lowest index from the set of colors previously assigned to other vertices is assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.
5. *Repeat steps 3 and 4 until all the graph vertices are colored.* Finally, all graph vertices are marked with the use of k colors.

Let us suppose that the successive vertices: v_6, v_5, v_2, v_4, v_3 and v_1 have been randomly selected for graph G_4. According to the presented algorithm, element v_6 is marked with the first color. Then, vertex v_5, which is not a neighbor of v_6, is chosen and therefore it is also classified to the first color class. The subsequent vertex, i.e., v_2 is connected with v_5, by an edge, so the general number of colors is increased, and v_2 receives the second color. Vertex v_4 is a neighbor of both v_6 and v_2, which means that it cannot be marked with any of the previously used colors. Finally, it is added to the new color class, and the general number of colors is increased and equals k=3. The subsequent vertex is v_3. Since it is connected

by an edge with v_5, it cannot be classified to the first independent set. However, it is not a neighbor of v_2, and therefore it is finally marked with the second color. The last vertex, v_1, is connected by an edge with both v_6 and v_2, whereas it is not a neighbor of v_4, therefore it gets the third color. As a result of the coloring process, three compatibility classes have been formed: C_1=$\{v_5, v_6\}$, C_2=$\{v_2, v_3\}$, C_3=$\{v_1, v_4\}$. Figure 4.12 illustrates the final result for the random ordered coloring of graph G_4.

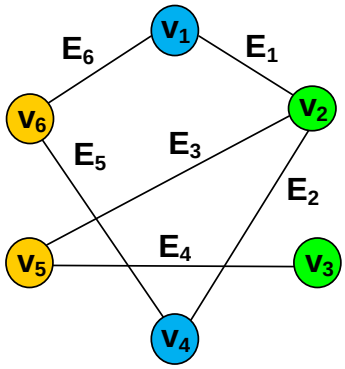

Fig. 4.12. Final result for the random ordered coloring of graph G_4

The result of the random ordered coloring method is similar to the one for the unordered coloring method. In both cases three color classes were obtained. However, the random ordered coloring caused a uniform distribution of vertices in all the compatibility classes which is essential for the further encoding of particular elements (see Chapter 6).

4.1.3.3. Largest-First Coloring

The Largest-First coloring algorithm of a graph is based on a sequential marking of all the vertices. However, unlike in the unordered method, elements from set V are selected according to the number of neighbors (i.e., the degrees of vertices). Therefore, vertices with the largest number of edges connecting them with other elements of set V are marked in the first place. The algorithm does not change the order during the whole coloring procedure, and thus the order of the vertices established at the beginning remains unchanged.

The LF coloring method may be divided into the following stages:

1. *Order the vertices by degree.* All elements from set V are sequenced from the vertex with the largest degree to the one with the smallest degree. If two vertices are of the same degree, the lexicographic order decides about the sequence.

2. *Reset the colors of all the vertices and establish the number of colors k=0.*

3. *Select the subsequent uncolored vertex v_i from set V, according to the already established order.* An uncolored vertex is chosen for the analysis (according to the sequence established in point 1).

4. *Assign the smallest possible color to vertex v_i.* If it is possible, the color with the lowest index from the set of colors already assigned to other vertices is assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.

5. *Repeat steps 3 and 4 until all the graph vertices are colored.* Finally, all the graph vertices are marked with the use of k colors.

For graph G_4, LF coloring is carried out as follows. In the first stage, the sequence of vertices is established. Since v_2 is of the highest degree (3), it is colored as the first one. Then, vertices v_1, v_4, v_5 and v_6, which are of the second degree, are marked. Vertex v_3, which is the neighbor of only one vertex, is colored as the last one. Finally, the ordered set of vertices selected successively for coloring with the LF method is as follows: $\{v_2, v_1, v_4, v_5, v_6, v_3\}$.

The coloring procedure itself is analogical to the unordered method. The subsequent vertices are marked with the lowest possible label among the already used colors. If there is no such possibility, the next color is added and value k is increased. At the beginning, vertex v_2 is colored with the first color, and variable k takes the value 1. In the next stage, vertex v_1 is selected for the analysis. It cannot be marked with the first color since it is a neighbor of vertex v_2. Therefore the general number of colors is increased, and v_1 gains the label with the second color. The next analyzed vertex is v_4. It is a neighbor of v_2, so it cannot be allocated to the first color class. However, the lack of mutual edge with v_1 allows for the assignment of the second color. Analogical coloring process takes place for vertex v_5. However, vertex v_6 is marked with the first color, because there is no edge connecting it with v_2. The last element from set V, which does not yet have a label, is vertex v_3. The vertex is not a neighbor of neither v_2 nor v_6, therefore the first color may be assigned to it. Eventually, the graph has been colored with k=2 colors. It means that one less color class has been used than in the case of unordered coloring. The result of LF ordered coloring is illustrated in Fig. 4.13.

4.1.3.4. Smallest-Last Ordered Coloring

The Smallest-Last graph coloring process is analogical to the LF coloring. Also in this case the vertices are sorted and then colored according to the established sequence. The basic difference is the process of ordering vertices. For SL method, vertices with the smallest degree are successively removed from the graph together with the edges incident to them. The operation is performed dynamically. The degrees of the remaining vertices are calculated in each phase. The procedure is repeated until all vertices and edges are removed from the graph. Finally, the vertices are ordered in the sequence reverse to the process of their reduction.

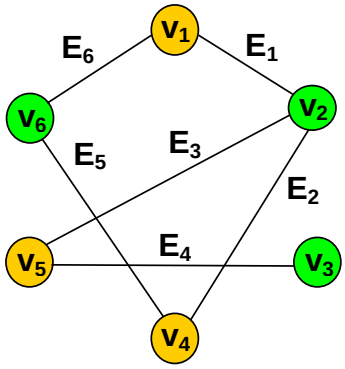

Fig. 4.13. Result of LF order coloring of graph G_4

For example, for graph G_4, the first reduced vertex is the one with the smallest degree, i.e., v_3. Also edge E_4, which is incident to it, is reduced. In the subsequent stage, the degrees of the vertices are calculated for the reduced graph, and then a vertex neighboring the smallest number of vertices is selected, i.e., v_5. It is removed from graph G_4 together with the edge incident to it, i.e., E_3. The presented process is repeated until the graph is reduced completely. In the subsequent stages the following vertices are reduced: v_1 (here, all the vertices are of the second degree so lexicographic order is applied), v_2, v_4 and then v_6. Finally, the vertices are ordered reversely to the reduction process, therefore the sequence of the coloring process is as follows: $\{v_6, v_4, v_2, v_1, v_5, v_3\}$.

The graph coloring is identical as for the earlier described methods. At the beginning vertex v_6 is marked with the first color. Then element v_4 is analyzed. Since it is a neighbor of v_6, it obtains the second color. The next vertex, i.e., v_2 is not connected by any edge with v_6, therefore it is marked with the first color. The procedure is performed for all the remaining vertices and finally the graph is colored with the use of two colors. Vertices v_2, v_3 and v_6 are marked with the first color, whereas - v_1, v_4 and v_5 with the second color. It means that the solution is identical to the one obtained for LF coloring algorithm (Fig. 4.13).

4.1.3.5. Graph Coloring with the Use of Exact Algorithms

The sequential methods of graph coloring, presented in the previous subchapters, do not always return the optimal solution. Sometimes it is necessary to obtain the smallest possible chromatic number. In practice it involves coloring of a graph with the use of the smallest possible number of colors. For this purpose, exact algorithms, which browse all possible variations of solutions, should be applied. One of most popular algorithms is the backtracking algorithm, searching recursively for the best solution. The method checks all possible solutions, selecting the best one, according to the posed criterion (e.g., the smallest number of colors).

In the case of graph G_4, vertex v_1 is colored with the first color in the exact method. Next, the algorithm marks recursively the subsequent vertices with the smallest possible color, analysing all possible combinations. Finally, the graph is colored with the use of two colors (vertices v_1, v_4 and v_5 are marked with the first color, whereas v_2, v_3 and v_6 - with the second color). The result is identical as in the case of LF and SL ordered coloring algorithms.

4.1.4. Determining Cliques of a Graph

According to Definition 3.3, a clique is a subset of graph vertices in which each two vertices are edge-connected. Determining all the maximal cliques of a graph is an NP-complete problem. It is impossible to find an exact solution to this problem in the polynomial time (Berge, 1973). This subchapter describes a method of determining cliques of a graph which is based on the backtracking algorithm. All possible solutions are checked, therefore the method makes sense only for relatively small graphs (the research has shown that the threshold value is approximately 20 vertices, for a larger number of vertices, the performance time increased considerably and the determination of the cliques took more than an hour).

It should be distinctly emphasized here that the algorithm determining cliques of a graph was developed and implemented only to supplement the basic methods of complementing discrete systems. The algorithm is not connected with the main objective of the book therefore some aspects of determining cliques of a graph are described very briefly (more information may be found in (Berge, 1973; Corno *et al.*, 1995)).

The developed method of clique determination is based on the following recursive algorithm:

1. *Set the value of the solution set $H=\emptyset$ and establish the sequence of vertices.* The initial set of solutions (set of all cliques) is reset, whereas vertices are sorted on the basis of their degrees, which enhances the chance for finding the biggest clique of a graph faster (Berge, 1973; Eiter and Gottlob, 1995; Corno *et al.*, 1995).

2. *Set the initial value of the clique set $Cl=\emptyset$.* Set Cl containing the vertices belonging to the clique is reset.

3. *Select the next vertex for the analysis.* At this stage, the next vertex v_i is selected according to the established sequence. If v_i does not belong to Cl, the dependency if set $Cl \cup v_i$ composes a clique is checked. If it does, v_i is attached to Cl. Otherwise, the vertex is neglected and the algorithm repeats performing point 3 for the subsequent vertex.

4. *Browse set H and reduce the excessive cliques.* It may happen that clique Cl overlaps one or several solutions found earlier. If any of the sets H is a subset of Cl, it is removed from H, whereas Cl is added to H. Analogically, if there is any Cl constituting a subset of any sets H, then the further analysis is neglected and the algorithm goes to point 2.

Steps 2-4 are repeated recursively so that all combinations of vertices are examined. The final solution is set H, which consists of set of cliques Cl_1,...,Cl_k. It is worth mentioning that the obtained solution includes maximal cliques, which results from the reduction of the excessive cliques in each cycle of the algorithm.

The obtained set H represents *a hypergraph*, whereas each clique is a hyperedge of the newly formed hypergraph. Hypergraph H contains n vertices and k edges. The number of vertices equals the number of vertices in the initial graph G, whereas the number of hyperedges k is less or equal to the number of edges m in graph G.

4.2. Mathematical Methods for the Study of Hypergraphs

The subchapter presents the most important mathematical algorithms investigating hypergraph structures. Each method is described in details and compared with the corresponding graph algorithm.

4.2.1. Hypergraph Complement

According to Definition 3.13, the complement of hypergraph H is hypergraph $\overline{H}$ containing the same vertices as H, whereas its set of edges comprises the complement of the set of edges of hypergraph H. Thus, in hypergraph $\overline{H}$ the vertices are adjacent if and only if there is no edge connecting them in hypergraph H. The determination of the hypergraph complement is an NP-complete problem (Berge, 1989). The analysis of the connections between the particular vertices is relatively uncomplicated, however, the determination of the relationship between the vertices in the newly-formed hypergraph is actually reduced to the determination of cliques in the complement. It is interesting that there is a wide spectrum of methods that can be used in the determination of a hypergraph complement. Obviously, the simplest and the most intuitive solution is to transform the hypergraph into a graph, next to determine its complement and then to find the cliques in this complement.

Another method is to determine the largest independent sets of the considered hypergraph. Such sets may be determined by the application of the Gentzen's theory (Tkacz, 2006) or by finding all the variants of hypergraph coloring (Kubale *et al.*, 2006). The final result are the maximal independent sets containing vertices which will form an edge in the target hypergraph $\overline{H}$.

There is also a relationship between an exact vertex cover and a hypergraph complement. The vertices comprised in the exact transversal of hypergraph H form simultaneously a hyperedge in the complement $\overline{H}$. Generally, the determination of all the exact transversals does not mean finding all the hyperedges in $\overline{H}$, although it may improve the process considerably. Determination of the complement in a c-exact hypergraph is particularly interesting. In this case, the edges in hypergraph $\overline{H}$ refer to exact transversals of hypergraph H, each of which may be determined in the polynomial time. Unfortunately, the problem with the number of all transversals, which can be exponential, still remains (Eiter, 1994).

4.2.2. Hypergraph Transversals

According to Definition 3.15, the presented hypergraph transversal (see previous chapter) is a set of vertices incident to each hypergraph edge. For example, the transversal of hypergraph H_2, incidence matrix of which is presented in Fig. 4.14, can be a set of 3 vertices: $T_1=\{v_1, v_3, v_6\}$. The particular vertices from set T_1 are incident to each edge of the hypergraph: v_1 is incident to edge E_1 and E_3, v_3 to E_5, whereas v_6 belongs to hyperedges E_2, E_3 and E_4. It should be emphasized here that the presented vertex cover is just one of the possible solutions. For example, the sets: $T_2=\{v_3, v_5, v_6\}$, $T_3=\{v_4, v_6\}$, etc., may also be transversals for hypergraph H_2.

$$A_2 = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \end{matrix}$$

Fig. 4.14. Incidence matrix A_2 for hypergraph H_2

It can be easily concluded from Fig. 4.14 that the smallest hypergraph transversal will consist of a two-element set. There is a single smallest vertex cover which consists of vertices v_4 and v_6, thus $\tau_1=\{v_4, v_6\}$.

The problem of the determination of the smallest hypergraph transversal is defined as an NP-complete problem (Berge, 1973; Eiter and Gottlob, 1995; Eiter and Gottlob, 2002). There is a number of algorithms of finding the smallest vertex cover of a hypergraph (Eiter and Gottlob, 2002; Wahlström, 2004; Kavvadias and Stavropoulos, 1999). The subsequent subchapters present the basic methods of determining the smallest hypergraph transversal. The presented algorithms have been illustrated with an example.

4.2.2.1. A Fast Reduction Algorithm

A fast reduction method is a modified version of an algorithm proposed by R. Rudell. In his doctoral dissertation, he presented the idea of reducing the table of simple implications in the process of minimization of logic functions (Rudell, 1989), which today constitutes an integral part of numerous methods and optimization systems (e.g., *Espresso*).

A fast reduction algorithm is based on the use of dependencies and relationships stored in a hypergraph. The first step to be done is to present a hypergraph in the form of an incidence matrix. The rows of the incidence matrix correspond to the hypergraph edges which should be covered with vertices represented by the

matrix columns. A fast reduction algorithm operates on the following hypergraph properties (Rudell, 1989; DeMicheli, 1994; Łuba, 2005):

- ⋆ *Essential column (essential vertex)* - according to Definition 3.19, if for a given vertex (column) there is a row with a single "one", it means that only this particular vertex belongs to a given hyperedge. Such vertices (columns) must be a part of each cover.

- ⋆ *Elimination of dominated columns* - a column dominates another column when elements of the previous column are greater than or equal to the elements of the subsequent one. Analogically, a vertex dominates another vertex if it is incident to at least all the edges which are also incident to another vertex. Dominated columns (vertices) may be neglected in the further considerations.

- ⋆ *Elimination of dominating rows* - a row dominates another row when the elements of the previous one are greater than or equal to the corresponding elements of the subsequent one. Analogically, an edge dominates another edge if it is incident to at least all vertices incident to another edge. Dominating rows (edges) may be neglected since each dominated set cover is a cover of a complete set (DeMicheli, 1994).

The determination process of the smallest transversal is based on the application of the above rules, while the phase of determining and removing the dominated columns and dominating rows is performed in cycles until the smallest vertex cover is achieved. The final result is obtained from the reduced matrix - the vertices which are incident to any edge of the reduced hypergraph will compose the searched smallest transversal.

In order to illustrate a fast reduction algorithm, the smallest transversal of hypergraph H_2 is determined. In the incidence matrix of the examined hypergraph, there exists an essential column. It represents vertex v_6 which, as the only one, is incident to edge E_4. It means that vertex v_6 has to be the part of each transversal of the hypergraph. Next, according to the presented algorithm, dominated columns are removed from matrix A_2. At this stage column 1 (dominated by column 2) and column 5 (dominated by 4) are reduced. In the next step, dominating vertices are reduced. For the considered example, rows 2 and 3 are identical, therefore one of them can be removed. It should be stated here that there is no difference which row is reduced (unlike in the case of reducing identical columns, it is certain that the hypergraph has more than one smallest transversal).

Figure 4.15 presents incidence matrix A_2' after the first reduction cycle. In the next stage, dominated columns are removed again (a column representing vertex v_2, since it is dominated by a column responding to vertex v_4). Next, dominating vertices are reduced (in the presented example, they are vertex responding to edges E_1 and E_2). At this stage, incidence matrix A_2'' contains 3 columns and 2 vertices (Fig. 4.16).

$$A_2' = \begin{array}{c} \begin{matrix} v_2 & v_3 & v_4 & v_6 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \\ E_4 \\ E_5 \end{matrix}$$

Fig. 4.15. Incidence matrix A_2 after the first reduction cycle

Subsequent reductions will not result in any changes, which means that matrix has been reduced. Fig. 4.16 shows that the smallest transversal consists of two vertices: $\tau_1=\{v_4, v_6\}$.

$$A_2'' = \begin{array}{c} \begin{matrix} v_1 & v_4 & v_6 \end{matrix} \\ \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_4 \\ E_5 \end{matrix}$$

Fig. 4.16. Reduced incidence matrix A_2

The fast reduction algorithm finds the approximate solution, which means that the determined result cannot be optimal. Admittedly, the found transversal is always minimal (contains no other transversals) but, in the given hypergraph, there may exist a more advantageous solution consisting of fewer number of vertices. Such a problem does not occur in the case of exact algorithms (e.g. backtracking method), but their exponential computational complexity is disadvantageous, and in some particular cases, the solution cannot be found at all.

4.2.2.2. A Backtracking Algorithm

A backtracking algorithm operates analogically to the method of searching for the graph cover presented in 4.1.2.1. First, a vertex marked with the smallest value (according to the lexicographic order) is selected and reduced. When the remaining vertex set is still a hypergraph cover, and the obtained result is better than the previously found, the current transversal becomes the best one. Next, the algorithm recursively carries out the process of searching for the best solution for all the vertices.

For hypergraph H_2, the vertex set consists of six elements: $V=\{v_1, \ldots, v_6\}$. According to the presented algorithm, vertex v_1 is selected to be reduced as the first one. After the reduction, the hypergraph consists of five vertices: $V^1=\{v_2, \ldots, v_6\}$. Since the remaining elements still form the cover, the set becomes the currently best found solution. Then vertex v_2 is removed, thus the vertex set is reduced to $V^2=\{v_3, \ldots, v_6\}$, which is also a cover and the smallest obtained transversal.

Similar situation takes place in the subsequent stage. The reduction of vertex v_3 limits the set of the remaining vertices (V^3=$\{v_4, v_5, v_6\}$), and the obtained solution still forms the cover of the hypergraph. In turn, the removal of vertex v_4 results in the fact that the vertex set is now limited to two vertices: V^4=$\{v_5, v_6\}$, which do not form the hypergraph cover, though. Therefore, the algorithm recursively returns to solution V^3, and now the subsequent vertex v_5 is removed from the set. The remaining set of vertices V^5=$\{v_4, v_6\}$ is also the smallest currently found transversal. The presented methodology is repeated for all the hypergraph vertices, and the subsequent stages are performed analogically to the ones presented above. Finally, set V^5 appears to be the best transversal. It means that the found solution is identical to the one obtained in the case of searching for the cover with the use of the fast reduction algorithm.

4.2.2.3. A Greedy Algorithm

The method for the determination of transversals presented in the previous subchapter, based on the backtracking algorithm, always returns the best solution. The disadvantage of the method is its exponential computational complexity, which means that given problem may not be solved, or the obtained solution may not be reached in the satisfying time.

As in the case of graph covers, the greedy method searches for the solution on the basis of a locally optimal decision. A great advantage of the algorithm is its speed, whereas its disadvantage - the fact that there is no possibility to determine if the found solution is really the best one.

The principle of operation of a greedy algorithm is practically identical as for the graph cover (4.1.2.2). The method also searches for the solution on the basis of the local optimum, therefore in the subsequent stages, a vertex with the smallest degree is selected for the analysis needs. The essential difference is the fact that the method assumes that there exist hyperedges which may connect more than two vertices. The method searching for transversals is as follows:

1. *Determine the value of the cover set T=$\emptyset$.* At the beginning, a set of covers (transversals) is empty, vertices will be determined and added in the subsequent cycles.

2. *Select an essential vertex, and if such a vertex does not exist, select a vertex with the largest degree.* At this stage, the algorithm selects an optimal local solution. An essential vertex (which has to be a part of the searched cover) is selected. If such a vertex does not exist, a vertex incident to the largest number of hyperedges is selected for the further analysis. If there exist more than one solution (e.g., there exist two essential vertices, or two vertices are of the same degree), the algorithm selects a vertex randomly.

3. *Add the selected vertex to the cover set T, and next remove the vertex from the hypergraph together with all the edges which are incident to it.*

4. *Check if T is a transversal, if not, repeat the algorithm from step No. 2.* In this step, set T is checked. If the vertices belonging to the set form the searched cover, the algorithm finishes its operation. Otherwise, the method proceeds to the next cycle in which the next local optimum (i.e., the subsequent essential vertex of the largest degree) is searched.

The algorithm starts its operation with the determination of the value of set $T=\emptyset$. Next, an essential vertex is selected in order to find the local optimum. For hypergraph H_2, it is vertex v_6, since it is the only one which is incident to column E_4. Therefore it is added to set T and simultaneously removed from the hypergraph together with the edges incident to it (i.e., E_2, E_3 and E_4). Thus, the hypergraph has been reduced to five vertices and two hyperedges. Figure 4.17 presents incidence matrix $A_2^{'''}$ of hypergraph H_2 after the reduction of vertex v_6.

$$A_2^{'''} = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \left[\begin{array}{ccccc} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \end{matrix}$$

Fig. 4.17. Incidence matrix A_2 after the reduction of vertex v_6

Next, the condition if set T constitutes a transversal of hypergraph H_2 is examined. Since the relation is not satisfied, the cycle is repeated. Among the remaining vertices, there exists one vertex v_4 incident to two hyperedges, whereas the remaining vertices are of the first degree. It means that v_4 is added to T, and next it is removed from the hypergraph. Since set T realizes vertex cover, the algorithm is finished.

In the considered example, the solution is $T=\{v_4, v_6\}$. This is simultaneously the smallest cover, and the identical result was obtained in the case of searching for a transversal using the exact methods. At this point it needs to be emphasised that the solution was already found in the second cycle. It clearly demonstrates the predominance of the greedy method over the earlier presented algorithms, if the computational complexity and the time of search are the criteria. Unfortunately, as it was mentioned before, the method does not return the best solution. The further part of the chapter considers the example in which a greedy algorithm finds a transversal, which is not the smallest one, though. An exemplary hypergraph H_3 with incidence matrix A_3, demonstrated in Fig. 4.18, will be used for the analysis.

The presented hypergraph has $|V|=7$ vertices and $|E|=12$ edges. There exists one smallest transversal. The set consists of three vertices: $\tau=\{v_1, v_2, v_3\}$. Such a solution may be obtained in both the fast reduction algorithm and in the backtracking method. However, the result will be different for the greedy algorithm, which is demonstrated below.

In hypergraph H_3 there is no essential vertex, therefore a vertex of the largest degree is selected in the first cycle of the greedy method. In this case, it is vertex v_4, which is incident to six edges: E_1, E_3, E_5, E_7, E_9, E_{11}. The vertex is added to

$$A_3 = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \\ E_8 \\ E_9 \\ E_{10} \\ E_{11} \\ E_{12} \end{matrix}$$

Fig. 4.18. Incidence matrix A_3 of hypergraph H_3

the searched set of solutions T, whereas the hypergraph is reduced. Both vertex v_4 and all the hyperedges incident to it are removed. The resultant structure has the incidence matrix illustrated in Fig. 4.19.

$$A_3' = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_5 & v_6 & v_7 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{matrix} E_2 \\ E_4 \\ E_6 \\ E_8 \\ E_{10} \\ E_{12} \end{matrix}$$

Fig. 4.19. Incidence matrix A_3 after the reduction of vertex v_4

Since no cover of the initial hypergraph H_3 has been found, the greedy algorithm repeats the procedure of vertex selection. Again, there is no essential vertex, therefore a vertex with the largest degree is selected for the subsequent considerations. It results from Fig. 4.19 that vertex v_5 should be selected, since it is incident to the largest number of hyperedges of all the remaining vertices. Therefore, vertex v_5 is added to set T, whereas edges E_2, E_4 and E_6 are removed from the hypergraph. Since the searched transversal is still unknown, the greedy algorithm repeats the procedure for the hypergraph reduced by vertices v_4 and v_5. The matrix of the reduced hypergraph is presented in Fig. 4.20. The considered structure includes the remaining three edges and five vertices.

$$A_3'' = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_6 & v_7 \end{matrix} \\ \left[\begin{matrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{matrix}\right] \end{array} \begin{matrix} E_8 \\ E_{10} \\ E_{12} \end{matrix}$$

Fig. 4.20. Incidence matrix A_3 after the reduction of vertices v_4 and v_5

In the subsequent stage, one of the three vertices: v_3, v_6 or v_7, will be reduced at random. Thus, it is possible to obtain several solutions. In each case, one more cycle will have to be performed, since set T will still not constitute the cover of hypergraph H_3. It will happen no sooner than in the subsequent reduction of the hypergraph vertices. Independently from the vertex selected to be removed (v_3, v_6 or v_7), set T will finally consist of four elements. The example presented below considers all the possible solutions:

- ⋆ If vertex v_3 is selected, in the next step there will remain vertices v_2 or v_7 to choose, therefore the transversal may consist of vertices T_1=$\{v_4, v_5, v_3, v_2\}$ or T_2=$\{v_4, v_5, v_3, v_7\}$;

- ⋆ If vertex v_6 is selected, in the next stage there will remain vertices v_2 or v_7 to choose, so the transversal may consist of vertices T_3=$\{v_4, v_5, v_6, v_2\}$ or T_4=$\{v_4, v_5, v_6, v_7\}$;

- ⋆ If vertex v_7 is selected, in the subsequent stage there will remain vertices v_3 or v_6 to choose, thus the transversal may consist of vertices T_5=T_2=$\{v_4, v_5, v_7, v_3\}$ or T_6=T_4=$\{v_4, v_5, v_6, v_7\}$.

The above considerations demonstrate that the greedy algorithm will return a set consisting of four vertices. In each case it will be the minimal transversal of the hypergraph (that is the one which contains no other transversal), however, it will not be the smallest cover. It results from the method itself, which makes decisions exclusively on the basis of local information. This is true in the case of the selection of vertex v_4, which on the one hand is incident to the largest number of edges, but on the other hand its selection made it impossible to achieve the global minimum, which in the presented example is the smallest transversal consisting of three elements: τ_1=$\{v_1, v_2, v_3\}$.

It should be emphasised here that the selection of the proper algorithm determines the way in which discrete systems are designed and analyzed. The application of the particular methods is conditioned by the given criteria, and the obtained results oscillate between the exactness of the searched solution and the time of its realization.

4.2.3. Exact Transversals of a C-Exact Hypergraph

A c-exact hypergraph is a very interesting structure in the class of hypergraphs. Its basic feature is the fact that each pair of compatible vertices is a component of at least one exact transversal. The most essential advantage of a c-exact hypergraph is the polynomial computational complexity in the case of determination of the subsequent exact transversals, which is shown in Chapter 5. It is a very important property in comparison to the remaining classes of hypergraphs (i.e., the ones which are not exact), for which the determination of exact transversals is connected with the exponential computational complexity (Berge, 1989; Eiter, 1994; Elbassioni and Rauf, 2010).

Obviously, such an essential reduction of the computational complexity results from the fact that the considered hypergraphs are strictly limited to the class of exact hypergraphs. Nevertheless, the research carried out by the author of this work confirms the practical application of exact hypergraphs primarily in both designing and analyzing digital systems. This book discusses an innovative way of the decomposition of a discrete system described with Petri nets through searching for subsequent transversals in concurrency hypergraphs. The decomposition methods used so far are based on approximate (greedy) algorithms and the application of exact methods was connected with exponential computational complexity. Chapter 6 presents the concept based on the utilization of exact transversals in the process of the decomposition of Petri nets into subnets of an automaton type.

This chapter describes the author's own idea of determining subsequent transversals in a c-exact hypergraph. For the needs of the proposed method, the Dancing Links (DLX) algorithm was adopted and modified. The algorithm was described in (Knuth, 2000) who proposed a solution to the exact cover problem which uses a four-direction list. The links between the particular elements of the list are dynamically removed and recovered (hence the name "dancing links"). Such an approach considerably reduces the time necessary to obtain the subsequent exact cover sets, therefore the algorithm is widely used in solving various problems (e.g., placing queens on a chessboard, or solving sudoku). Taking into account the properties of a c-exact hypergraph (maximal independent sets correspond to the searched exact transversals), the DLX algorithm has been suited for the needs of the determination of exact transversals in a c-exact hypergraph. The devised method has been included in the realized hypergraph library and constitutes an integral part of *Hippo* system.

As it is shown below, the simplified algorithm, searching for the subsequent exact transversals was later realized in C++:

1. If the list describing a c-exact hypergraph is empty, the solution has been found. Return the current solution. Otherwise select edge E_e containing the smallest number of vertices.

2. For each vertex v_i comprised in E_e perform the following operations:

   (a) add vertex v_i to the current cover (an exact transversal);

(b) cover vertex v_i together with all hyperedges to which v_i is incident and all the vertices belonging to the edges;

(c) perform recursively point (2a);

(d) uncover vertex v_i together with all edges incident to it and all vertices belonging to these edges;

(e) remove vertex v_i from the current cover (an exact transversal).

The most essential element of the described algorithm is the option of *covering* and *uncovering* particular objects of the four-list. The process is reduced to the change of the values of indicators of particular objects (physically they are neither added to nor removed from the list), which significantly speeds up the whole cycle of finding the transversals. The detailed description of the DLX method can be found in (Knuth, 2000).

For the needs of the methods developed in this book, the algorithm has been improved so that the searching process can be halted when the found set of the exact transversals covers all hypergraph vertices. Thus there is no need to determine all exact transversals in a given exact hypergraph. If the first and each subsequent transversal can be determined in the polynomial time, the whole number of all exact transversals may be exponential (Eiter, 1994).

4.2.4. Vertex Coloring of a Hypergraph

Vertex coloring of a hypergraph (further referred to as coloring) is strictly connected with coloring of the graph vertices. The idea behind it is the same: all vertices of a hypergraph must be marked in such a way that any vertices linked with an edge were of a different color. The subsequent subchapters present some methods of hypergraph coloring (Berge, 1973; Lovász, 1973; Franklin and Saluja, 1994; Krivelevich and Sudakov, 1998; McDiarmid, 1997; Tuza and Voloshin, 2000; Tuza *et al.*, 2002; Acharya and Acharya, 2003; Czumaj and Sohler, 2005; Kubale *et al.*, 2006). Due to the fact that there are significant analogies between the algorithms of graph hypergraph coloring, only most essential features of particular methods will be discussed.

4.2.4.1. Unordered Sequential Coloring

The procedure of unordered sequential coloring of hypergraphs proceeds as follows:

1. *Reset the colors of all the vertices and set the number of colors k=0.* At the beginning the colors of all the vertices are reset. Moreover, variable k, storing the number of currently used colors, is also reset.

2. *Select a subsequent uncolored vertex v_i from set V.* An uncolored vertex is selected for the analysis (according to the lexicographical order).

3. *Assign the smallest possible color to vertex v_i.* If it is possible, the color with the smallest index in the set of colors already assigned to other vertices, is

assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.

4. *Repeat steps 2 and 3 until all the hypergraph vertices are colored.* Finally, all vertices of the hypergraph are marked with k colors.

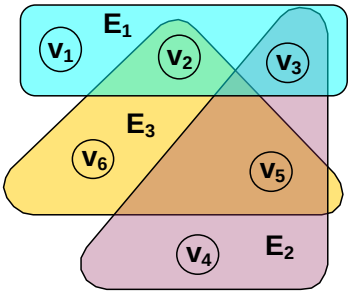

Fig. 4.21. Hypergraph H_4

For the exemplary hypergraph H_4, presented in Fig. 4.21, coloring proceeds as follows: vertex v_1 is selected for the analysis as the first one. The element is marked with the first color. The vertex cannot be colored with the same label as it is linked to v_1 with a hyperedge. Therefore, v_2 takes the second color. A similar situation takes place for vertex v_3, which is adjacent to both v_1 and v_2. Therefore the vertex is marked with the third color. The next examined element is v_4. The vertex can be colored with the first label, since there is no edge connecting it to v_1. Vertex v_5 cannot be marked with neither the first (adjacent to v_4) nor the second (adjacent to v_2), or the third (adjacent to v_3) color. Therefore the fourth color is assigned to it. The last vertex (v_6) is neither adjacent to v_1 nor to v_4, thus it may be marked with the first label. The final result of the unordered coloring is presented in Fig. 4.22.

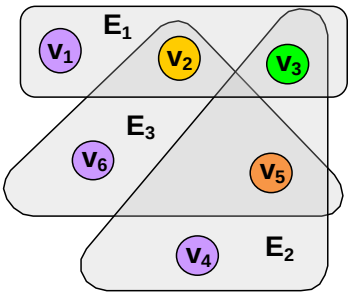

Fig. 4.22. Unordered coloring of hypergraph H_4

4.2.4.2. Randomly Ordered Coloring

The randomly ordered coloring method includes the following stages:

1. *Order the vertices randomly.* All the elements from set V are ordered in a random sequence.

2. *Reset the colors of all vertices and set the initial number of colors k=0.*

3. *Select a subsequent uncolored vertex v_i from set V, according to the earlier determined order.* An uncolored vertex is selected for the analysis (according to the order determined in point 1).

4. *Assign the smallest possible color to vertex v_i.* If possible, the color with the smallest index from the set of the already assigned colors is assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.

5. *Repeat steps 3 and 4 until all the hypergraph vertices are colored.* Finally, all the hypergraph vertices are marked with the use of k colors.

Let us assume, that for hypergraph H_4 the subsequent vertices have been randomly selected: v_4, v_2, v_6, v_1, v_3 and v_5. In this case, element v_4 is marked with the first color. Next, vertex v_2 is analyzed. Since it is not adjacent to vertex v_4, it also takes the first color. Vertex v_6 is incident to the same edge as v_2, therefore it has to be marked with the next label - the second color. A similar situation takes place for v_1, which is adjacent to v_2. Since there is no edge connecting v_1 and v_6, the vertex is finally marked with the second color. For element v_3 it is necessary to increase the number of colors, since it is adjacent to both v_4 (the first color) and v_1 (the second color). The last vertex (v_5) can be colored with none of the previous labels, therefore it is marked with the fourth color. Figure 4.23 illustrates the final result of the randomly ordered coloring of hypergraph H_4.

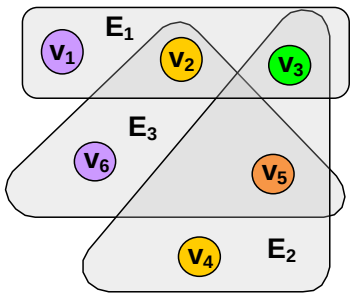

Fig. 4.23. Coloring of hypergraph H_4 with randomly ordered vertices

4.2.4.3. Largest-First Ordered Coloring

The method of Largest-First ordered coloring proceeded as follows:

1. *Order the hypergraph vertices according to the degrees.* All elements from set V are sequenced starting from the vertex with the highest degree down to the vertex with the lowest degree. If two vertices are of the same degree, the order is determined on the basis of the lexicographic order.

2. *Reset colors of all the vertices and set the number of colors k=0.*

3. *Select the subsequent uncolored vertex v_i from set V, according to the previously determined order.* An uncolored vertex (according to the order determined in point 1), is selected for the analysis.

4. *Assign the smallest possible color to vertex v_i.* If it is possible, a color with the smallest index from the set of already assigned colors, is assigned to vertex v_i. Otherwise, vertex v_i is marked with a new color, and value k is increased by one.

5. *Repeat steps 3 and 4 until all the hypergraph vertices are colored.* Finally, all the vertices of the hypergraph are marked with the use of k colors.

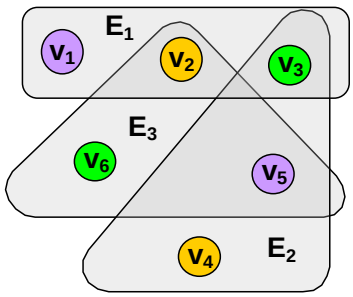

Fig. 4.24. LF ordered coloring of hypergraph H_4

In hypergraph H_4, vertices v_2, v_3 and v_5 are of the third degree, whereas v_1, v_4 and v_6 - of the second degree. Therefore the LF ordering will take the form: $\{v_2,v_3,v_5,v_1, v_4,v_6\}$, and it will result in the occurrence of three color classes k=3. Vertices v_2 and v_4 will be marked with the first color, v_3 and v_6 with the second color, whereas v_1 and v_5 - with the third color (Fig. 4.24).

4.2.4.4. SL Ordered Coloring of a Hypergraph

The SL ordered coloring of a hypergraph is analogical to the LF coloring. Vertices are sorted and then colored according to the previously determined order. The ordering proceeds as follows: the subsequent vertices of the lowest degree, together

with the edges that are incident to them, are removed from the hypergraph (in each phase the degrees of the remaining vertices are calculated). The procedure is finished when all the vertices and edges are removed from the hypergraph. Finally, the vertices are ordered in the reverse order to the reduction process.

For hypergraph H_4, vertex v_1, the one of the lowest degree is reduced as the first one (although vertices v_4 and v_6 are also of the same degree, the lexicographic order is the decisive factor). In the subsequent steps, the vertices are reduced in the following order: v_2, v_6, v_3, v_4 and v_5. Therefore finally the vertices are ordered as follows: $\{v_5, v_4, v_3, v_6, v_2, v_1\}$. Figure 4.25 presents hypergraph H_4, vertices of which are colored according to SL ordered algorithm.

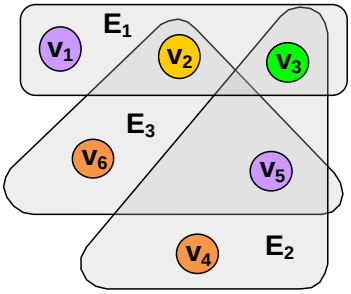

Fig. 4.25. SL ordered coloring of hypergraph H_4

4.2.4.5. Coloring of a Hypergraph with the Use of Exact Algorithms

Similarly to graphs, hypergraphs may also be colored with the use of exact algorithms. The procedure is identical: all possible solutions are examined and the best result is selected. Thus, the algorithm omits any possible local minimums, returning the most beneficial solution. However, for its exponential complexity, the operation time is considerably longer than for methods searching for approximate results (in numerous cases, the result is even impossible to obtain, which is shown in Chapter 7).

For hypergraph H_4, the result of coloring with the exact method is identical to the one obtained for coloring with the use of LF-ordering. It appears that the best solution consists of three colors, and each color class contains two vertices.

4.3. The Analysis of the Relations between Graphs and Hypergraphs

This chapter presents the analysis of the relations between graphs and hypergraphs. All the properties mentioned in the book are the author's conclusions and thoughts which occurred during the research and work over the two structures. Some of the described features have been commonly known for a couple of years

(Lovász, 1972; Berge, 1973; DeMicheli, 1994). It seems that the current literature does not present any clear systematisation of graphs and hypergraphs. What is more, it neither provides any comparison between them nor any determination of their similarities and differences. The chapter is a preliminary attempt to face the problem, and the presented conclusions will be the base for the characteristics of the proper methods of discreet systems decomposition, described in Chapter 6.

4.3.1. General Relations between a Graph and a Hypergraph

The most obvious difference between the discussed structures is the number of vertices which may be linked by an edge. In a graph, an edge may contain maximally two vertices, whereas in a hypergraph, edges may be incident to an arbitrary number of vertices. It implies that *each graph is simultaneously a hypergraph, whereas the inverse property is not true.* The rule is of a great importance for algorithms operating on the discussed structures: methods based on hypergraphs also refer to graphs. Naturally, in numerous cases, the property is of an insignificant importance, but e.g., the coloring algorithms intended for hypergraphs are also efficient for traditional undirected graphs.

4.3.2. Relations between Graph and Hypergraph Vertex Covers

Despite a great similarity between a graph vertex cover and a hypergraph vertex cover, they do not return the same results. It is due to the properties of the structures. The fact that an edge can connect maximum two vertices, significantly limits the vertex cover in a graph. For example, Fig. 4.26 presents graph G_5 consisting of only three vertices. As can be easily noticed, G_5 is a full graph since all its vertices are connected with each other. However, the smallest cover consists of two vertices, which results from the fact that each edge is not incident to one vertex.

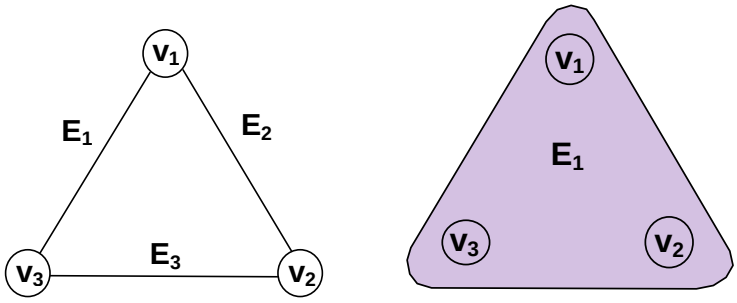

Fig. 4.26. Graph G_5 and hypergraph H_5

The smallest transversal of hypergraph H_5, corresponding to the structure of graph G_5, consists of just one element. This is so as one edge connects all vertices. Therefore the vertex cover of a graph is not equivalent to the vertex cover of

a hypergraph. Obviously, there are some exceptions in which such a situation may take place, but generally, the result of a graph cover may not be substituted with an equivalent hypergraph cover and vice versa.

The extension of a graph cover is a clique cover of a graph (Berge, 1973). The method consists in finding the smallest cover with the use of the largest full subgraphs (cliques). Formally, a clique cover is constituted of sets V_1,V_2,..., V_k, which are subsets of set V consisting of all the vertices of graph G such that each set V_i contains a full subgraph of the full graph G. Additionally, for each edge $\langle u, v\rangle \in E$ there exist such set V_i which contains both u and v. In practice, a hypergraph transversal is equivalent to the clique cover of a graph. **It should be clearly stated here that the determination of the clique cover of a graph is always preceded by the stage of the determination of cliques in a graph, which is an NP-complete problem. The step is not required for hypergraphs, since cliques (i.e., hyperedges) compose an integral property of a hypergraph.**

4.3.3. Relations between Graph and Hypergraph Coloring

Vertex coloring of graphs and hypergraphs is very strictly interrelated. The problem of hypergraph coloring is the generalization of graph coloring (Kubale *et al.*, 2006; Lovász, 1973). It results from the reciprocal relations between the two structures. If there is a connection between two vertices in a given graph, they cannot be marked with the same color. Analogical situation takes place in a corresponding hypergraph, irrespectively to the fact that the edge connecting the two mentioned vertices can also be incident to other vertices of the hypergraph. It means that problems solved with the graph coloring may be equivalently calculated with the use of hypergraph coloring. The relation was used in this book and served as a base for the research and experiments carried out to testify the effectiveness of the developed methods for the decomposition of discrete systems.

Due to the properties of hypergraphs and the stored information about cliques which determines the connections between particular vertices, the algorithm of hypergraph coloring should be slightly faster than the corresponding graph algorithm. It should be clearly emphasized that the reduction does not refer to the order of computational complexity, since the determination of the best solution still implies the exponential computational complexity.

4.4. Summary

The chapter presents the most important mathematical methods for the study of undirected graphs. It presents graph algorithms which enable the determination of the complements of the structures. Moreover, it describes a method enabling the determination of all the cliques in an undirected graph which, in practice, correspond with edges in the newly formed hypergraph. The chapter contains a detailed description of the algorithms of vertex cover which, in the case of hypergraphs, have particularly essential properties since the determination of subsequent ex-

act transversals is connected with the reduction of the computational complexity from exponential (in a general case) to polynomial (for c-exact hypergraphs). It describes five coloring algorithms of graphs and hypergraphs. Because of the fact that there is a considerable analogy in the process of vertex coloring, the methods for graph and hypergraph coloring are very similar, whereby the hypergraph coloring may be slightly faster due to the properties of the structures (the results of the research including the comparison between the coloring of graphs and concurrency hypergraphs are presented in Chapter 7).

All described algorithms have been practically realized and implemented in the C++ programming language. A measurable result of the developed methods is the *Hippo* system whose functionality is described in Appendix A.

Chapter 5

DEFINITIONS AND THEOREMS

This chapter constitutes a set of definitions and theorems of the author introduced in the book. The following notions are determined: *concurrency hypergraphs*, *sequentiality hypergraphs* and *selection hypergraph*. The theorems improving mathematical operations which exploit the mentioned structures are also introduced.

All introduced theorems and definitions refer to live and safe dynamic Petri nets that belong to Marked Graph class. The discussed mathematical apparatuses have been practically used in the decomposition processes of discrete systems (Chapter 6).

Definition 5.1. ***Concurrency hypergraph*** *is a hypergraph that presents a concurrency relation between the places in a Petri net. The vertices of the concurrency hypergraph correspond to various places in a Petri net, whereas hyperedges determine relations between these places. Vertices in a concurrency hypergraph are connected by a hyperedge if and only if they are marked in the same state of a marking graph. A concurrency hypergraph is* ***proper*** *if none of its hyperedges is contained in any other hyperedges.*

A concurrency hypergraph is a mathematical structure useful for the analysis of the properties of Petri nets. The subsequent edges of the concurrency hypergraph may be obtained directly from an initial Petri net in a polynomial time (Wiśniewski *et al.*, 2012), however the number of hyperedges can be exponent. What is important, it can be also obtained in a different way, i.e. directly from a reachability graph (Wiśniewska *et al.*, 2007b).

The work focuses on the issues connected with the decomposition of Petri nets into state machine components (decomposition methods are described in Chapter 6). The detailed description of the application of a concurrency hypergraph in the analysis as well as the theorems connected with the correctness of the constructed Petri net may be found in (Wiśniewska and Adamski, 2006).

Figure 5.1 presents an incidence matrix of a concurrency hypergraph for net MN_1 presented in Fig. 2.3. The hypergraph vertices reflect macroplaces of the considered macronet. The edges, in turn, determine sets of places marked in the same state which may be performed by an automaton in parallel. For example, vertices M_2, M_3 and M_4 are connected by an edge, which means that the places are concurrent.

Theorem 5.1. ***A concurrency hypergraph of a Petri net is a c-exact hypergraph.***

$$A_{W_{MN_1}} = \begin{array}{c} \begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 \end{matrix} \\ \left[\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{matrix}\right] \end{array} \begin{matrix} \\ E_1 \\ E_2 \\ E_3 \\ E_4 \end{matrix}$$

Fig. 5.1. Incidence matrix of a concurrency hypergraph for macronet MN_1

Proof. According to Definition 3.21, a c-exact hypergraph is a hypergraph in which each pair of vertices not connected by an edge is incorporated in at least one exact transversal. With respect to a Perti net, the relation is understood as follows: each two places in a Petri net (belonged to Marked Graphs) which are not linked by the concurrency relation are the components of at least one automaton subnet. If two places of a Petri net are not in a concurrency relation, they must be performed sequentially, since both relations are mutually exclusive and complementary (Murata, 1989). It results from the fact that both places must belong to at least one SMC in which processes are performed sequentially. □

Theorem 5.2. ***A state machine component in a Petri net corresponds to an exact transversal in the concurrency hypergraph.***

Proof. A state machine component (SMC) is a set of places which are performed sequentially (Murata, 1989). SMCs form a closed cycle between the marked places in a Petri net. Therefore each SMC in a concurrency hypergraph is represented as a transversal (an intersection of all edges of a hypergraph). Additionally, all places comprised in SMC must be performed sequentially which means that they must have exactly one representative in each global (concurrent) state. It results in the fact that each subnet (SMC) is simultaneously an exact transversal of a concurrency hypergraph. □

Definition 5.2. ***A sequentiality hypergraph*** *is a hypergraph obtained as a result of the determination of exact transversals in a concurrency hypergraph. The vertices of a sequentiality hypergraph represent places (or macroplaces) of a Petri net, whereas the edges correspond to the determined exact transversals. Each edge of a sequentiality hypergraph represents a single subnet (SMC) in which particular places of a Petri net are in a sequentiality relation, i.e., they are not active at the same time.*

A sequentiality hypergraph determines SMCs which occur in a Petri net that belong to Marked Graph class. Fig. 5.2 ppresents an incidence matrix of a sequentiality hypergraph for macronet MN_1. There are five exact transversals in the macronet, therefore the sequentiality hypergraph contains five edges. Each edge determines a single sequential automaton (SMC).

$$A_{S_{MN_1}} = \begin{array}{c} \begin{matrix} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 \end{matrix} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{matrix}$$

Fig. 5.2. Incidence matrix of a sequentiality hypergraph for macornet MN_1

In practice, it is frequently necessary to perform the operation of selection in order to reject excessive components and to obtain the best solution. For this reason, *a selection hypergraph* is determined.

Definition 5.3. ***A selection hypergraph*** *is a hypergraph obtained in the reduction of edges in a hypergraph dual to a sequentiality hypergraph. The vertices of a selection hypergraph refer to the exact transversals of a concurrency hypergraph, whereas the edges determine the places of a Petri net after the reduction.*

The process of creating a selection hypergraph may be divided into two essential stages. First, a hypergraph dual to a sequentiality hypergraph should be determined. In practice it reduces to the determination of a transposed matrix for a given sequentiality hypergraph. In such a matrix, it is necessary to remove the dominant rows (corresponding to the edges of a dual sequentiality hypergraph), that is, the ones whose elements (values) are greater or equal to the corresponding elements of another arbitrary row. Next, dominated columns are removed from the hypergraph (i.e., vertices of the hypergraph). The resulting hypergraph is a searched selection hypergraph. The reduction operation and the whole process of the decomposition of Petri nets is discussed in details in Subchapter 6.1.

The propositions of theorems and sketches of proofs connected with the computational complexity of algorithms operating on exact hypergraphs are presented below. The work focuses on presenting induction proofs. The detailed and formal mathematical proofs definitely exceed the scope of this work and may constitute a broad area for further scientific research. It is worth mentioning that all theorems have been verified empirically with the use of the set of real test modules, i.e., live and safe Petri nets that belong to the class of Marked Graph.

Basic notions connected with the computational complexity of algorithms (Papadimitriou, 1994) will be determined for the sake of theorems. As already mentioned, all considered issues are connected with the time complexity of an algorithm, determined as a relation between the size of input data and the number of basic operations performed by the algorithm (Aho *et al.*, 1974). The process of the analysis of the computational complexity has mainly been taken from (Eiter, 1994), but also from (Rudell, 1989) and (DeMicheli, 1994).

In the analysis, the following notations will be used: n - number of hypergraph

vertices, m - number of hyperedges. The worst possible solution to a given problem is taken into account in all the considerations (Aho *et al.*, 1974).

Theorem 5.3. ***The first exact transversal in a c-exact hypergraph may be determined in a polynomial time.***

Proof. Let us consider the following algorithm $FIRST\_EXACT$ searching for first exact transversal D_1 in hypergraph H of n vertices and m edges:

1. *Select an arbitrary vertex v_i of a c-exact hypergraph and add it to set D_1.*
2. *Remove vertex v_i and all the hyperedges to which v_i is incident and all the vertices belonging to these edges.*
3. *If the hypergraph is empty (contains no edges or vertices) - the solution has been found. Otherwise perform point 1 recursively.*

In terms of computational complexity, a single cycle of the above algorithm is the analysis of point 2. All edges incident to v_i are removed in maximum m steps. In turn, the removal of all vertices comprised in theses edges is connected with $O(n)$ complexity. Therefore, a single cycle of the algorithm is of a quadratic computational complexity $O(m * n)$, in the case of a primary hypergraph. Naturally, in each subsequent recursive call, the complexity of a single cycle may be of at least the same, quadratic, order.

As a result of the operations carried out in point 2, the following are removed from the hypergraph vertex v_i, all edges to which it is incident, as well as all vertices which belong to these hyperedges. It means the removal of all the vertices remaining in an incompatibility relation with vertex v_i from the hypergraph. Thus, only the vertices compatible with vertex v_i remain in the hypergraph. Let us consider all the possible results of point 2 of the algorithm:

1. *The reduced hypergraph is empty.*

   A solution has been found (an exact transversal, determined by set D_1).

2. *One vertex or all vertices of the hypergraph remain in a compatibility relation.*

   According to the definition of a c-exact hypergraph, each pair of adjacent vertices forms at least one exact transversal. If a single vertex remains after the reduction process and is adjacent to v_i, it constitutes a complement to the exact transversal (in the subsequent recursive step it is added to set D, the subsequent reduction results in the fact that the hypergraph is empty and the algorithm finishes its operation). Similar situation takes place when all remaining vertices in the hypergraph are compatible. Then, the subsequent vertices are recursively added until the hypergraph is completely reduced (at most in $n - 1$ subsequent recursive calls).

3. *All vertices in the reduced hypergraph remain in the incompatibility relation.*

Such a state means that independently from the fact which vertex is selected in the subsequent step of recurrence, the hypergraph is completely reduced, and the solution is found.

4. *In the hypergraph, there are at least three vertices left, which are both in the compatibility and incompatibility relation.*

   Previous steps exclude the possibility when one or two vertices (mutually compatible or incompatible) remain in the hypergraph. What is more, such a situation necessitates that at least two vertices of the hypergraph are compatible and two are incompatible. To illustrate the discussed case, Fig. 5.3 presents a matrix of hypergraph H_D, containing three vertices v_1, v_2, v_3. Vertices v_1 and v_2 remain in the incompatibility relation, whereas pairs v_1 and v_3 as well as v_2 and v_3 are compatible. Since all vertices were compatible to vertex v_i before the reduction, the operation could not modify the relations between them in any way (only vertices incompatible to v_i and edges which could not be incident to the remaining vertices were removed from the primary hypergraph). Therefore, the obtained reduced hypergraph must still remain a c-exact hypergraph. The algorithm is recursively continued until the hypergraph is completely reduced (at most in $n-1$ steps).

$$A_D = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 \end{matrix} \\ \left[\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} E_1 \\ E_2 \end{matrix}$$

Fig. 5.3. Matrix A_D of exact hypergraph H_D

It results from the above considerations that the algorithm performs, in the worst case, n recursive calls (it is possible if all vertices of a c-exact hypergraph remain in the compatibility relation). It is very important that independently from the selected vertex, the subsequent steps result in obtaining exact transversals. Taking into account the fact that each single cycle (a single recursive call) is realized in $O(m * n)$, time, the whole algorithm may be of a maximum cubic complexity $O(m*n^2)$. The fact means that the determination of the first transversal in a c-exact hypergraph is connected with at most polynomial computational complexity (which may even be linear in some particular cases - e.g., when all vertices are compatible). □

Theorem 5.4. ***Subsequent exact transversals of a c-exact hypergraph may be selected in a polynomial time.***

Proof. Let us expand the previously presented algorithm in such a way that it searches for all exact transversals in hypergraph H of n vertices:

1. *Select edge E_e, containing the smallest number of vertices.*

2. *For each vertex v_i comprised in edge E_e*:

   (a) *Add vertex v_i to the currently searched transversal D_d.*

   (b) *Remove vertex v_i together with all hyperedges to which v_i is incident and all vertices belonging to these edges.*

   (c) *If the hypergraph is empty (contains no edges or vertices) - exact transversal D_d, has been found, and is added to the searched set of all transversals D.*
   Otherwise, perform recursively point 1.

The discussed algorithm ALL_EXACT is a modified version of the *X-algorithm*, proposed by D.E. Knuth for a general search for an exact cover in an arbitrary hypergraph (Knuth, 2000). The general outline and the effective way of the implementation of the algorithm with the use of "dancing links" (DLX) is presented in Subchapter 4.2.3.

It is shown below that the algorithm finds the subsequent exact transversal in a hypergraph in the polynomial time (generally, for an arbitrary hypergraph, it is an exponential complexity (Knuth, 2000)). The notion "subsequent" refers here to the determination of the next solution on the basis of the currently found exact transversal (i.e., after finding the first exact transversal in a hypergraph, the second is found in the polynomial time, after the second exact transversal has been found - finding the third - also involves the polynomial computational complexity, etc.). It is worth stressing that if the determination of each subsequent exact transversal is connected with polynomial complexity, the number of exact transversals itself may be exponential, which is shown in (Eiter, 1994). Therefore, there is no algorithm determining all exact transversals in polynomial time.

Let us examine the algorithm in terms of computational complexity. The method functions as follows: first edge E_e (containing the smallest number of vertices) of a c-exact hypergraph is selected for the analysis. The edge of the smallest degree may be determined in the maximal time $O(m * n)$, with the assumption that the hypergraph is fully examined. In practice, the operation is frequently performed linearly (Knuth, 2000). Next, for each vertex v_i contained in edge E_e, the operations presented in 2 are performed subsequently and independently. The maximal number of vertices composing a single edge equals n, and for the subsequent recursive calls, the value must be smaller. Stages from 2a to 2c are performed identically as for determining the first exact transversal. The difference occurs for a recursive call in which once again an edge with the smallest number of vertices is selected (the previous algorithm involved a selection of an arbitrary vertex). Let us consider two possibilities:

1. *Selected set of solutions D is empty (no transversal has been found yet).*

   In this case, the algorithm performs the subsequent operations analogically to the algorithm $FIRST\_EXACT$. Additionally, in each cycle (of recurrence), edge E_e of the smallest degree is selected. For the further analysis, the first vertex v_i (no matter which) being the component of edge E_e is selected. The algorithm recursively performs subsequent operations until the

hypergraph is completely reduced, which in turn means that the first exact transversal has been found. The additional stage of the selection of edge E_e does not influence the increase in the order of the computational complexity of a single cycle in the method, which was also quadratic in the case of the $FIRST\_EXACT$ algorithm. Thus, finding the first exact transversal in a c-exact hypergraph with the use of the ALL_EXACT algorithm is connected with the complexity $O(m * n^2)$ at most.

2. *The searched set of solutions contains one or more exact transversals.*

   Such a state means that the first (or the subsequent) exact transversal has been found in the hypergraph. The analyzed hypergraph is currently empty, therefore the algorithm recursively returns to the previous cycle (dominant recursive call). If in this cycle the analyzed edge E_e contained one vertex - the algorithm proceeds to the subsequent dominant recursive cycle (and the vertex added in this recursive cycle is removed from the current transversal). Otherwise, the next vertex v_j, being a component of edge E_e is selected for the analysis (it is obvious that the maximum number of returns of the algorithm to the higher level of recurrence equals n, i.e., the number of all vertices of the hypergraph, and that it is the particular case when all vertices are compatible). Next, transversal D_d is completed with vertex v_j, and the vertex itself is removed from the hypergraph together with all the edges it is a component of and all the vertices incident to these edges. It means that all vertices which are in the incompatibility relation with vertex v_j are removed from the hypergraph, and all the remaining vertices have to be compatible to v_j. Such a state, in turn, means a situation analogical to the one considered in point 1, where the exact transversal was searched. The analyzed algorithm recursively performs the subsequent cycles until the hypergraph is reduced, which means that the subsequent exact transversal has been found. In a particular case it may happen that the found exact transversal differs from the previous one only in one element (interchangeably: vertices v_i and v_j). The analysis reveals that the computational complexity of the determination of the second exact transversal is of the same order as in the case of the first exact transversal. The subsequent returns to the higher levels of recurrence are performed until edge E_e contains more than one vertex (n - the maximum number of returns). Finding the subsequent exact transversal, together with vertex v_j may be maximally connected with the complexity $O(m * n^2)$ in the case when the algorithm returns to the first level of recurrence.

The presented solutions show that the determination of each subsequent exact transversal in a c-exact hypergraph may be performed in $O(m * n^2)$ time. It is worth stressing that the identical computational complexity was obtained by T. Eiter for a hypergraph of exact transversals. □

In order to make the presented sketches of proofs more clear, the process of the analysis of the computational complexity is illustrated on the following example. There is hypergraph $H_{W_{MN_1}}$, the incidence matrix of which is presented

in Fig. 5.1. Hipergraf $H_{W_{MN_1}}$ is a proper concurrency hypergraph of a Petri net. Therefore, by virtue of Theorem 5.1 it is simultaneously a c-exact hypergraph.

Initially, the set of exact transversals D is empty. In its first step, the algorithm searches for an edge containing the smallest number of vertices. Let us assume that the full review of hyperedges is performed (the number of vertices which belong to a given edge is calculated each time). In this case, the determination of the edges of the smallest degree is performed in the quadratic time $O(m * n)$. For the further analysis, edge E_1 is selected, since it contains just a single vertex M_1. The vertex is added to the currently searched transversal D_d. All edges incident to the vertex are removed from the hypergraph as well as all vertices belonging to the edges (in this case only edge E_1 is removed). The reduction in this case means the removal of one vertex and one edge, therefore it also does not increase the computational complexity, which still remains quadratic. The incidence matrix A_{W1} of a hypergraph formed as a result of the reduction of vertex M_1 and a hyperedge E_1 is shown in Fig. 5.4. Compared to the original hypergraph, H_{W1} contains $n-1$ vertices and $m-1$ edges.

$$A_{W1} = \begin{array}{c} \begin{array}{cccccc} M_2 & M_3 & M_4 & M_5 & M_6 & M_7 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} E_2 \\ E_3 \\ E_4 \end{array}$$

Fig. 5.4. Incidence matrix of hypergraph H_{W1}

In the next step the algorithm recursively proceeds to point 1 and performs operations on the reduced hypergraph. Once again, an edge (E_4) containing the smallest number of vertices is determined, which is performed in time $O((m-1) * (n-1))$. This time the edge contains two vertices: M_2 and M_7. The process of vertex selection is of an insignificant importance. For the greater clarity, vertex M_2 is selected according to the lexicographic order. The vertex is added to the searched transversal D_d, and next all the edges incident to M_2 are removed together with the vertices which belong to these edges. In this case, it involves the removal of $m-1$ edges and $n-1$ vertices (quadratic complexity), and the reduced hypergraph is empty. It means that the first exact transversal has been found and that it contains elements: $D_1{=}D_d{=}\{M_1, M_2\}$. To sum up, the determination of the first exact transversal in the considered hypergraph, taking into account the lexicographic order of vertices, is connected with the quadratic complexity. Obviously, another example (even with a different order of vertices) could result in a different complexity, but it is clearly visible that in the worst case the computational complexity of the determination of the first exact transversal in a hypergraph is at most of a cubic order.

The subsequent stage is the return of the algorithm to a higher level of recurrence. Simultaneously, vertex D_d is removed from set M_2 (since it is not an

element of the searched transversal at this stage of recurrence), and the subsequent vertex belonging to edge E_4, i.e., M_7, is selected for the analysis. The vertex is added to the searched transversal, thereby complementing M_1. As a result of the reduction of vertex M_7, edge E_4 is removed from the hypergraph together with vertex v_2. Hypergraph H_{W2}, whose incidence matrix is shown in Fig. 5.5, is the result of performed operations.

$$A_{W2} = \begin{array}{c} \begin{matrix} M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \left[\begin{matrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{matrix} \right] \end{array} \begin{matrix} E_2 \\ E_3 \end{matrix}$$

Fig. 5.5. Incidence matrix of hypergraph H_{W2}

The further analysis of the algorithm is obvious. In the next stage, edge E_2 is selected, and vertex M_3 is added to the transversal. Hyperedge E_2 is removed from the hypergraph which results in the fact that vertices M_5 and M_6 remain in the reduced hypergraph. According to the lexicographic order, vertex M_5 is selected in the next recursive call, and after its reduction the hypergraph is empty. Thus, the subsequent transversal D_2=$\{M_1,M_7,M_3,M_5\}$ has been found. It can be easily noticed that despite the greater number of recursive calls, the complexity of the algorithm has remained polynomial. The subsequent exact transversal are determined analogically. The algorithm recursively returns to higher levels and searches for subsequent unreduced vertices. For example, transversal D_3 will be found already in the subsequent cycle, through replacing vertex M_5 with M_6: D_3=$\{M_1,M_7,M_3,M_6\}$. The two remaining exact transversals may be determined in a similar way. Analogically, the two remaining exact transversals: D_4=$\{M_1,M_7,M_4,M_5\}$ and D_5=$\{M_1,M_7,M_4,M_6\}$ may be determined.

Chapter 6

METHODS FOR DECOMPOSITION OF DISCRETE SYSTEMS

The chapter presents innovative methods for decomposition of discrete systems. The developed algorithms constitute a complete tool enabling the decomposition of a concurrent automaton into sequential automatons whereby a discrete system may be easily realized with the use of real digital systems, e.g., FPGA (Maxfield, 2004).

The first of the discussed methods is connected with the decomposition of a concurrent automaton described by Petri nets. An innovative manner of dividing the controller into state machine components (SMCs) has been proposed. In the realization, exact transversals of the concurrency hypergraph have been applied so that the determination of the subsequent automaton subnets involves a polynomial computational complexity.

Another proposed method enables the decomposition of the discrete system memory. The algorithm is grounded on the reduction of the microinstruction size. The result of the performed operations is the decomposition of the primary memory into a reduced part (the memory with encoded microinstructions) and a decoder part. The algorithm is based on the already exiting solutions. However, the introduction of hypergraphs enables its improvement through the application of an arbitrary known method of the determination of the smallest transversal in a hypergraph. The memory reduction method shown in this chapter is a peculiar complement to the method for decomposition of the controlling part of a discrete system. As a result, a complete method enabling the division of a discrete system has been obtained (for both the controlling part and the memory parts of particular sequential automata). However, each of the discussed algorithms may be applied independently.

6.1. Parallel Decomposition of a Discrete System

This subchapter presents an innovative method for the parallel decomposition of concurrent systems, described with the use of Petri nets, carried out through the decomposition of hypergraphs. The aim of the decomposition is the division of a reconfigurable logic controller into concurrent modules each of which can be then optimized and synthesized with the use of the classical theory of digital automata.

In contrast to the solutions used so far, the author's entirely new method for the determination of non-concurrent sets is based on the

computation of the subsequent exact transversals in a concurrency hypergraph. Thanks to this, the subsequent subnets are determined in a polynomial time. Such an approach is impossible for traditional concurrency graphs, in which the determination of independent sets (frequently based on coloring or finding a clique cover) involves exponential computational complexity.

6.1.1. Problem Formulation

When designing a discrete system (in particular the part realizing a function of a controller), it may appear that the size of the system exceeds the frames imposed by the size of a prototyped system. Then, it is divided into a series of collaborating subsystems. The decomposition facilitates the designing process and its results may be used in effective encoding of internal local states of a controlling system (Brayton *et al.*, 1984; Sentovich *et al.*, 1992; Kania, 2004; Łuba, 2005). The most frequently applied method of a digital system synthesis is the decomposition of a concurrent automaton into sequential component automata (Papachristou, 1979; Girault and Valk, 2003; Kozłowski *et al.*, 1995; Adamski and Węgrzyn, 2009).

The method for the decomposition of Petri nets with the use of concurrency graph coloring or searching for clique covers for the complement of a concurrency graph (i.e., a non-concurrency graph, or sequentiality graph) has already been known (Banaszak *et al.*, 1993; Adamski, 1990; Karatkevich, 2007). This book includes only the most important features and properties which are indispensable for the proposed method to be presented. The detailed description of the transformation of a net into a macronet as well as its decomposition with the use of graphs may be found in (Valette, 1978; Adamski *et al.*, 2005).

The previous method for decomposition of logic controllers with the use of Petri nets may be divided into the following stages:

1. *Formation of a macronet for the considered Petri net.* At this stage, it is necessary to determine a macronet which is a condensed version of a given Petri net, maintaining its basic structure and properties. The aim is to deprive the net of all the fragments which do not influence its final solution and which increase the time of analysis. The formed macromodules contain multi-input/output transitions as well as places replacing singly-marked fragments of the net.

2. *Determination of a reachability graph.* The activity involves the analysis of changes in the net markings as the ready transitions are fired, and recording any possible states in a form of a graph. The graph vertices contain the set of marked places in a given state, the edges correspond to the fired transitions (Valette, 1978; Banaszak *et al.*, 1993; Adamski and Chodań, 2000).

3. *Determination of a concurrency graph.* At this stage the concurrency graph is determined. The graph stores the information about relations between the places of the decomposed Petri net, and it constitutes the basis for the

determination of concurrency relations (Kozłowski *et al.*, 1995; Zakrevskij *et al.*, 2002; Mielcarek, 2009).

4. *Concurrency graph coloring.* The stage involves the determination of the concurrency relations for the graph determined in the previous point. In general, the operation reduces to graph coloring (Kubale, 2002). Independent sets determined in such a way contain macroplaces which are components of particular SMCs (places marked with the same color form a single concurrent system).

5. *Replacing the macroplaces with suitable subsets of places of the considered net.* This is the last step in the decomposition of a controller into SMCs. Concurrent controllers obtained in such a way may be modelled with the use of languages of equipment description, and then implemented, e.g., in matrix systems FPGA (Maxfield, 2004; Wiśniewski, 2009).

For designers, the greatest problem and the weakest point of the algorithm presented above is the concurrency graph coloring. To determine the optimal solution it is necessary to repeat the coloring process, which involves the exponential computational complexity. For larger Petri nets the decomposition process may take very long or even be impossible at all. In order to eliminate the above problem, one of the approximate coloring methods, described in Subchapter (4.1.3, is interchangeably used).

In (Adamski and Wiśniewska, 2006; Wiśniewska, 2005; Wiśniewska *et al.*, 2006; Wiśniewska and Adamski, 2008a; Wiśniewska and Wiśniewski, 2010) numerous authors' algorithms can be found in which Petri nets are decomposed on the basis of hypergraph coloring. Such an approach allows the reduction of time which is necessary to perform the process of partition of a controller. However, finding an optimal solution still involves the application of the repetitive coloring and exponential complexity.

In the next subchapter, an entirely innovative approach to the decomposition of a controlling part of a discrete system is proposed. The partition is realized through the determination and selection of subsequent exact transversals in a concurrency hypergraph. The result of the process is the optimal partition of a controller, the same as for repetitive coloring of a graph or a concurrency hypergraph. The determination of the subsequent exact transversal in a c-exact hypergraph presenting concurrent relations in a Petri net is connected with the polynomial complexity, which is shown in Chapter 5.

6.1.2. The Idea of the Proposed Method

Developing a new parallel decomposition method of a controlling system, it was taken into account that each exact transversal of a concurrency hypergraph corresponds to a single SMC in a given Petri net that belong to the class Marked Graph. In the proposed algorithm, the subsequent exact transversals are determined, i.e., a controller can be decomposed into the subsequent concurrent automata. The operation is performed until all local states (vertices of a concurrency hypergraph)

are covered with the determined exact transversals. If the number of the determined automata is larger than the minimal number of SMCs covering the whole Petri net, it is necessary to perform the selection process. The process is also executed through the determination of exact transversals. For this purpose, a selection hypergraph is defined, and the exact transversals, determined in the hypergraph, exactly indicate which automaton components form the final solution of the decomposition of a Petri net.

Particular steps of the decomposition of a Petri net, based on the application of exact transversals of a c-exact hypergraph, are presented below. For the needs of the discussed algorithm, the author's own definitions of *a concurrency hypergraph*, *a sequentiality hypergraph* as well as *a selection hypergraph* introduced in Chapter 5 have been used.

1. *Formation of a macronet.* At this stage a macronet should be determined, i.e., a condensed version of a given Petri net. The stage proceeds analogically as for the standard method for the decomposition of a Petri net.

2. *Determination of a concurrency hypergraph.* A concurrency hypergraph represents true concurrency between the places of Petri net. The vertices of a concurrency hypergraph correspond to the places of a Petri net, whereas hyperedges determine relations between these places (Wiśniewska and Adamski, 2006; Wiśniewska *et al.*, 2007b). The concurrency hypergraph may be obtained directly form the initial Petri net (Wiśniewski *et al.*, 2012) or in a different way, i.e. from the reachability graph.

3. *Determination of subsequent exact transversals in a concurrency hypergraph.* In a concurrency hypergraph the subsequent exact transversals are determined according to the author's algorithm presented in details in Chapter 4. The process is performed until all vertices of the hypergraph are covered with the smallest possible number of the determined exact transversals (the value is already known at the stage of the determination of the concurrency hypergraph and equals to the degree of the largest hyperedge).

   If the number of found transversals equals to the number of the smallest possible automaton subnets, into which the primary Petri net can be decomposed, the decomposition process is finished. Otherwise, the selection operation has to be performed.

4. *Determination of a sequentiality hypergraph.* The vertices of a sequentiality hypergraph respond to the vertices of a concurrency hypergraph, whereas the edges - to the determined exact transversals. The sequentiality hypergraph illustrates places which are performed sequentially in a given Petri net. The detailed description of this structure and the analysis of the relation between the concurrency and sequentiality hypergraphs are presented in Chapter 5.

5. *Determination of a hypergraph dual to the sequentiality hypergraph.* At this stage, vertices and edges are interchanged, i.e., the incidence matrix of the sequentiality hypergraph is transposed.

The subsequent two stages are connected with the reduction of the hypergraph. The idea of the reduction has been taken from (Rudell, 1989), who proposed an innovative concept of the minimization of logic functions (in practice the algorithm was used in *Espresso* system). The idea of the reduction of the array of covers of prime implicants inspired the author to adapt an apply the method in the reduction of hypergraphs (in this case a hypergraph dual to the sequentiality hypergraph). The result of the performed reduction is a selection hypergraph, discussed in details in Chapter 5.

6. *Reduction of dominant edges in a hypergraph.* Dominant edges may be neglected as each cover of a dominated set is a cover of a full set (Rudell, 1989; DeMicheli, 1994). Through the application of this operation, the excessive relations (represented by hyperedges) which occur between the vertices (i.e., excessive connections between exact transversals in a concurrency hypergraph) are removed from the hypergraph. The process of reducing the dominant edges is connected with the quadratic computational complexity, which is shown in (Rudell, 1989).

7. *Reduction of dominated vertices in a hypergraph.* Analogically to the previous stage, dominated vertices are removed from the hypergraph (Rudell, 1989; DeMicheli, 1994). Also in this case, the reduction process involves the quadratic computational complexity (Rudell, 1989).

   The result of the conducted reductions is a selection hypergraph. The vertices of the selection hypergraph correspond to the subsequently found exact transversals in the concurrency hypergraph (i.e., with particular sequential automata), whereas the edges illustrate the relations between these transversals (between sequential automata). Therefore, the exact transversals can be applied once again. The smallest exact transversal in a selection hypergraph determines a set of vertices corresponding to the selected exact transversals of the concurrency hypergraph.

8. *Determination of the smallest exact transversal (if any exist) in a reduced dual selection hypergraph.* The smallest exact transversal of a selection hypergraph indicates the solution. The selected vertices of the selection hypergraph determine the searched exact transversals in a concurrency hypergraph. The sets of macroplaces forming part of each of the found exact transversals in the concurrency hypergraph constitute the searched sequential automata.

   This point does not apply if there is no exact transversal in a hypergraph. In such a case the simple smallest transversal should be found. Detailed description how to calculate the smallest transversal is presented in Subchapter 6.2.

   The determined SMCs refer to sequential automata that may then be designed with the use of any method of the synthesis of controlling systems (e.g., as a finite automaton of FSM states (Baranov, 1994)). Each of the

controllers may also be subjected to the further process of decomposition, depending on the assumed technology and the target digital system (Sentovich *et al.*, 1992; Sasao, 1999; Rawski *et al.*, 2003; Kania and Kulisz, 2007; Barkalov and Titarenko, 2009; Wiśniewski, 2009; Bukowiec, 2009; Łabiak, 2005; Doligalski and Adamski, 2010)

6.1.3. The Example of a Parallel Decomposition

In order to present the idea of designing a reconfigurable logic controller, an example taken from the literature illustrating the application of Petri nets to represent control systems, is presented below (Blanchard, 1979; Adamski, 1990; Roguska, 2001).

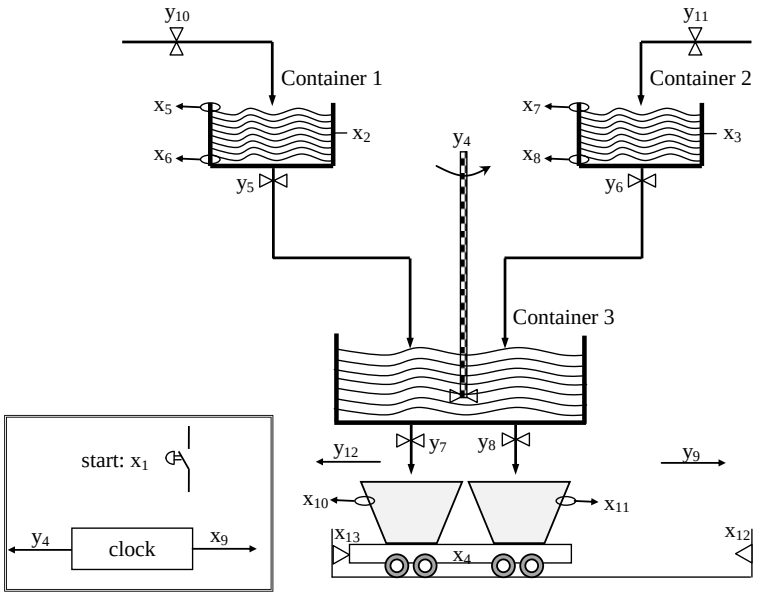

Fig. 6.1. Beverage production machine

Figure 6.1 presents a machine for beverage production and distribution (*Mixer*). The controller functions in the following way: pressing the start button (x_1) initiates the processes in which containers 1 and 2 are being filled and the cups for the beverages are delivered. Then valves y_{10} and y_{11} dare opened until the containers are filled, and this information is indicated respectively by sensors x_5 and x_7. In turn, the delivery of the cups is connected with the movement of the cart with cups (active signal y_{12}) and finishes when the cart reaches sensor x_{13}. While the containers are being filled, the ingredients are being prepared, which is indicated with sensors: x_2 - for the first container, x_3 - for the second container and x_4 - for the cart. The ready components are poured into the third container and mixed,

which is realized by opening valves y_5 and y_6 (active signal y_4). The valves are closed after emptying the containers 1 and 2 and mixing all the. The situation is signalled with sensors x_6, x_8 and x_9 respectively. When one of the containers is ready, it is independently filled and closed (signals x_{10} and x_{11}). After the completion of both processes, the cart with cups moves back to its initial position, which is signalled with signal y_9. Sensor x_{12} is active when the system is ready for the further operation.

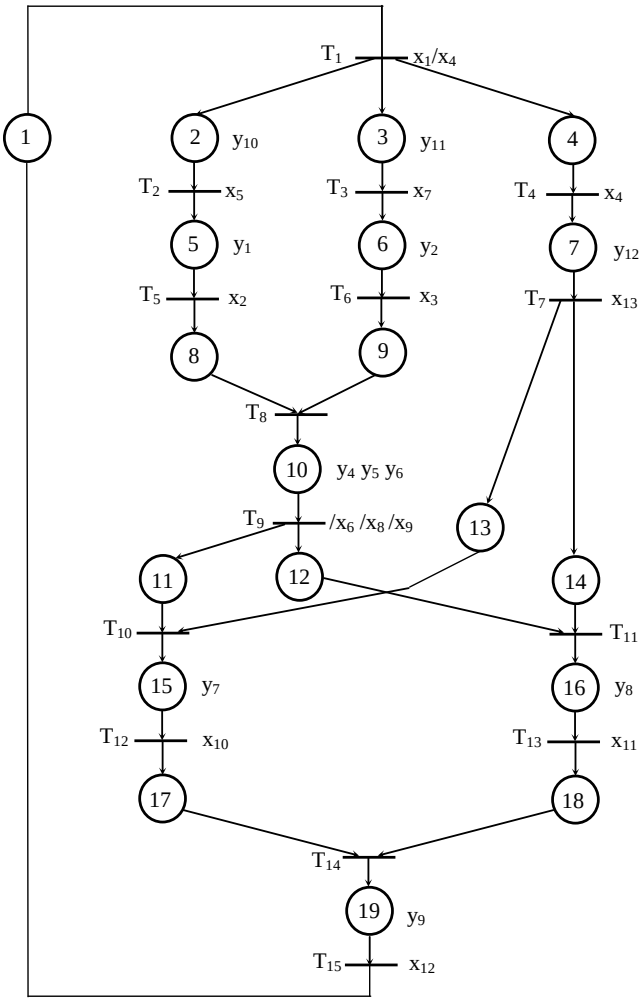

Fig. 6.2. Petri net PN_2 controlling the machine for beverage production

Formally, the presented controller consists of thirteen input and twelve output signals. Figure 6.2 presents Petri net PN_2 illustrating the operation of the machine for beverage production (Blanchard, 1979).

The first step in the decomposition of the controlling part of the discrete system, is the formation of a macronet for a given Petri net. This is a condensed version of a given Petri net (Valette, 1978), maintaining its basic features and properties. The objective of the operation is to remove any sequential fragments from the net, which do not influence its final solution in the process of the analysis, and whose presence only increases the decomposition time.

Figure 6.3 presents macronet MN_2 for Petri net PN_2. Its sequential fragments have not been replaced with macroplaces. The resultant macronet MN_2 is a condensed version of net PN_2, maintaining its properties and features. The process of reduction allowed a considerable decrease of the primary Petri net. Structure PN_2 consists of 19 places, whereas its reduced equivalent contains 11 macroplaces. The figure presents visually how places of primary net PN_2 form the particular macroplaces in macronet MN_2. For example, macroplace MP_1 contains places 2, 5, 8; macroplace MP_2 places 3, 6, 9, etc.

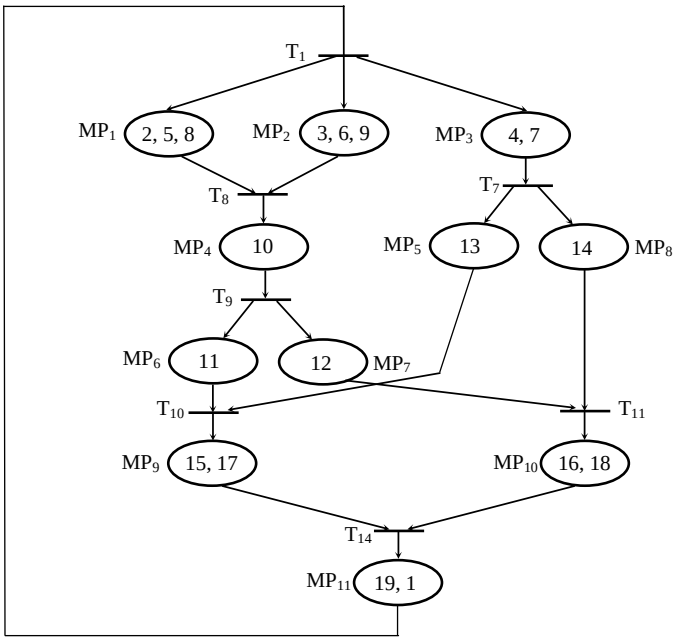

Fig. 6.3. Macronet MN_2 for Petri net PN_2

In the subsequent stage of the decomposition, a concurrency hypergraph is formed. This structure can be obtained in a various way. Here we will use additional formation of the reachability graph, however the concurrency hypergraph can be calculated directly from the Petri net (Wiśniewski *et al.*, 2012).

The reachability graph is determined on the basis of the primary Petri net or its condensed version, i.e., a macronet (the representation of a Petri net with a macronet allows considerable shortening of the analysis of concurrency, which

was proved in (Murata, 1989; Adamski *et al.*, 2005; Karatkevich, 2007)).

The formation process of a marking graph reduces to the analysis of changes in the net markings when the ready transitions are fired. The values are described in a form of a graph, whose vertices correspond to the set of places marked in a given state, whereas the edges determine the fired transitions.

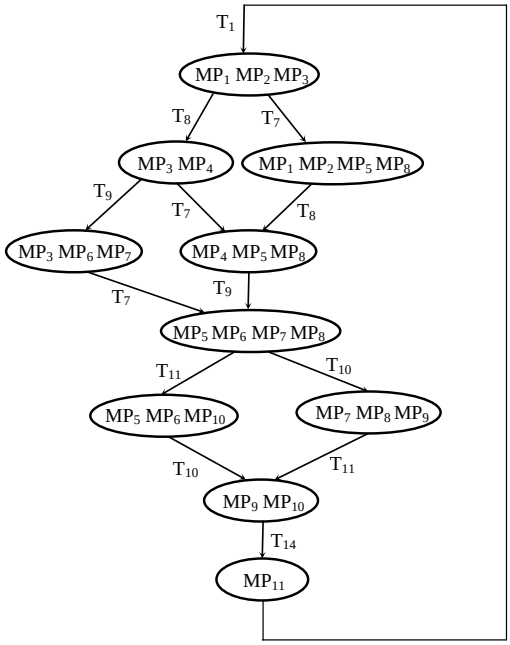

Fig. 6.4. Marking graph for macronet MN_2

Figure 6.4 presents a marking graph for macronet MN_2. The structure illustrates the basic relations in the considered automaton. The figure shows that, e.g., places MP_1, MP_2 and MP_3 are marked with the same state. It means that the three fragments are performed in parallel. Nevertheless, the thorough analysis of the concurrency relations between the particular states may be troublesome. Therefore, the process of the examination of the concurrency and decomposition of Petri net involves the application of a next step which is the determination of a concurrency hypergraph. The concurrency hypergraph perfectly captures the relations between the particular local states of the controlling part of a discrete system. The hypergraph vertices refer to the places of the analyzed Petri net, whereas the edges illustrate the relations between particular places of Petri net. What is essential, unlike the commonly used concurrency graph, the concurrency hypergraph holds full relations between particular states.

Figure 6.5 presents concurrency hypergraph H_{MN_2} for net MN_2. The hypergraph vertices reflect macroplaces in the considered macronet. The edges of the concurrency hypergraph determine the sets of places marked in the same state,

which may be performed by an automaton in parallel. For example, vertices MP_1, MP_2, MP_3 are linked by an edge, which means that the places are concurrent. The macronet shows that the macroplaces refer to the processes of preparing the containers and the ingredients, and they may be performed in parallel. Whereas macroplaces MP_1 (preparing the first ingredient) and MP_4 (mixing the ingredients) should be performed in a respective sequence. In fact, in the concurrency hypergraph, the places are not connected by a concurrency relation thus they are performed sequentially.

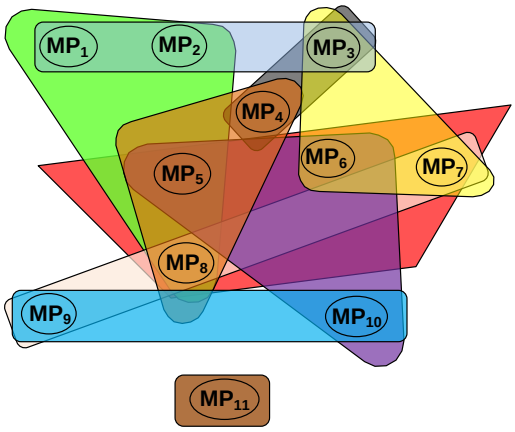

Fig. 6.5. Concurrency hypergraph H_{MN_2} for macronet MN_2

In order to illustrate the subsequent steps of the proposed idea of the decomposition of a controlling part of a discrete system better, the incidence matrix of concurrency hypergraph H_{MN_2} is additionally presented in Fig. 6.6.

The matrix columns refer to the macroplaces of the Petrie net, whereas the rows refer to the relations between the places. To increase the legibility of the matrix, the particular macroplaces in the figure have been called from M_1 to M_{11}.

At this stage, the smallest possible number of SMCs into which the Petri net can be decomposed is determined. The value defines the number of places which are performed concurrently. In the concurrency hypergraph, it equals the degree of the edge containing the largest number of vertices. In the analyzed example, hyperedges E_1 and E_3, degree of which equals 4, contain the largest numbers of vertices. Thus, Petri net PN_2 may be divided into at least four SMCs.

The next step is the determination of the subsequent exact transversals in the concurrency hypergraph. The process is performed until the found transversals cover all the vertices in the concurrency hypergraph.

$$A_{H_{MN_2}} = \begin{array}{c} \begin{array}{ccccccccccc} M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 & M_8 & M_9 & M_{10} & M_{11} \end{array} \\ \left[\begin{array}{ccccccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \\ E_6 \\ E_7 \\ E_8 \\ E_9 \\ E_{10} \end{array}$$

Fig. 6.6. Incidence matrix of concurrency hypergraph H_{MN_2}

In the presented example, six exact transversals are determined finally:

D_1=$\{MP_1, MP_4, MP_6, MP_9, MP_{11}\}$,
D_2=$\{MP_2, MP_4, MP_6, MP_9, MP_{11}\}$,
D_3=$\{MP_1, MP_4, MP_7, MP_{10}, MP_{11}\}$,
D_4=$\{MP_2, MP_4, MP_7, MP_{10}, MP_{11}\}$,
D_5=$\{MP_3, MP_5, MP_9, MP_{11}\}$,
D_6=$\{MP_3, MP_8, MP_{10}, MP_{11}\}$.

Since the number of the determined exact transversals covering all the vertices of the concurrency hypergraph is greater than the smallest possible number of SMCs, it is necessary to conduct the selection. Of all the determined exact transversals, an optimal solution should be selected for the further analysis, which would determine the smallest possible number of sequential automata (in this example - 4) into which the primary Petri net can be decomposed.

According to the presented decomposition algorithm, the sequentiality hypergraph is determined in the subsequent stage. The hypergraph vertices correspond to the found exact transversals, whereas the edges represent macroplaces of the Petri net.

Next, a hypergraph dual to the sequentiality hypergraph is determined. It means that hypergraph vertices and edges are interchanged. In practice, it means that the incidence matrix of the hypergraph is transposed. For the analyzed example, matrix $A_{H_D^*}$ of a hypergraph dual to the sequentiality hypergraph is presented in Fig. 6.7.

The next step is the reduction of dual hypergraph H_D^*. First, dominant edges (corresponding to rows of an incidence matrix) are removed from the hypergraph, i.e., the edges whose elements (values) are greater than or equal to the respective elements of another arbitrary edge MP_5. Then the hypergraph vertices are sub-

$$A_{H_D^*} = \begin{array}{c} \begin{matrix} D_1 & D_2 & D_3 & D_4 & D_5 & D_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ MP_1 \\ MP_2 \\ MP_3 \\ MP_4 \\ MP_5 \\ MP_6 \\ MP_7 \\ MP_8 \\ MP_9 \\ MP_{10} \\ MP_{11} \end{array}$$

Fig. 6.7. Incidence matrix of hypergraph H_D^* dual to the sequentiality hypergraph

jected to the analogical reduction process. The dominated vertices are removed from the final solution. The result of the performed operation is a selection hypergraph (Fig. 6.8).

$$A_{H_S} = \begin{array}{c} \begin{matrix} D_1 & D_2 & D_3 & D_4 & D_5 & D_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ MP_1 \\ MP_2 \\ MP_5 \\ MP_6 \\ MP_7 \\ MP_8 \end{array}$$

Fig. 6.8. Reduced hypergraph H_S

The last stage is connected with the determination of the smallest transversal in the reduced hypergraph H_S. Thus, a searched subset of exact transversals is determined. The operation is performed identically to the process of the determination of an exact transversal of the concurrency matrix, and for the subsequent considerations only solutions containing the smallest number of elements are taken into account. In hypergraph H_S two smallest exact transversals can be distinguished:

$\delta_1 = \{D_1, D_4, D_5, D_6\}$,
$\delta_2 = \{D_2, D_3, D_5, D_6\}$.

Both sets contain four elements each, therefore it does not matter which of them would be selected. In the considered example, the first solution is selected for the further analysis. It contains four selected exact transversals which determine sets of concurrent macroplaces (sequential automata). The number of exact transversals conforms to the previously established smallest number of concurrent automata net PN_2 may be decomposed into.

The first set contains places determined by transversal D_1. It consists of the following macroplaces: MP_1, MP_4, MP_6, MP_9 and MP_{11}. The second concurrent automaton is described by transversal D_4. Since its excessive places (which have already occurred in transversal D_1) are removed, it contains the following places: MP_2, MP_7, MP_{10}. Analogically, the third independent set consists of places MP_3 and MP_5, whereas the fourth set includes macroplace MP_8.

On the basis of the obtained results, the Petri net is divided into k=4 concurrent automata: a, b, c, d. Figure 6.9 presents the final result of the decomposition. If a given set of non-concurrent places does not contain the initially marked place of the net, a resting place containing a sign is added to it (e.g., place P_{20}).

The above example was verified experimentally. The controller was decomposed with the use of the classical method (backtracking coloring) as well as the one proposed in the book. The results of the experiments confirm the very high efficiency of the method. The time necessary to find the solution with the classical method was 18 times longer than when exact transversals were applied, and both algorithms returned identical solution. The detailed results of the research (including *the Mixer* net) are discussed in Appendix B.

Then the functional verification of the obtained results was carried out. For this purpose, three different versions of a controller were prepared to be implemented in a digital system, according to the description presented in Subchapter 2.2.2. The developed prototypes were modelled with the use of Verilog equipment (Brown and Vernesic, 2000; Thomas and Moorby, 2002). Next, a behavioral verification (simulation) of the particular versions was performed and the synthesis and implementation were conducted in a real system FPGA (system XC3S500E of *Spartan* family, made by *Xilinx*). The source codes for all the three models are placed in Appendix C. It should be distinctly emphasized here that the range of the work does not include the implementation of the discrete systems, the focus has been put on the presentation of new, more effective decomposition methods. The methods of the realization of Petri nets, and, above all, the decomposed sequential automata in digital systems provide a broad stream of research discussed, among others, in (Bilinski *et al.*, 1994; Kozłowski *et al.*, 1995; Fernandes *et al.*, 1997; Girault and Valk, 2003; Adamski and Węgrzyn, 2009; Banaszak *et al.*, 2008; Szpyrka, 2008).

First of the models constituted a description of an appliance realized as a Petri net with the use of "one-hot" encoding. Each of the states was encoded independently, and the number of flip-flops closely depended on the number of local states. In this case 19 flip-flops were obtained, which is equivalent to the number of states of a Petri net.

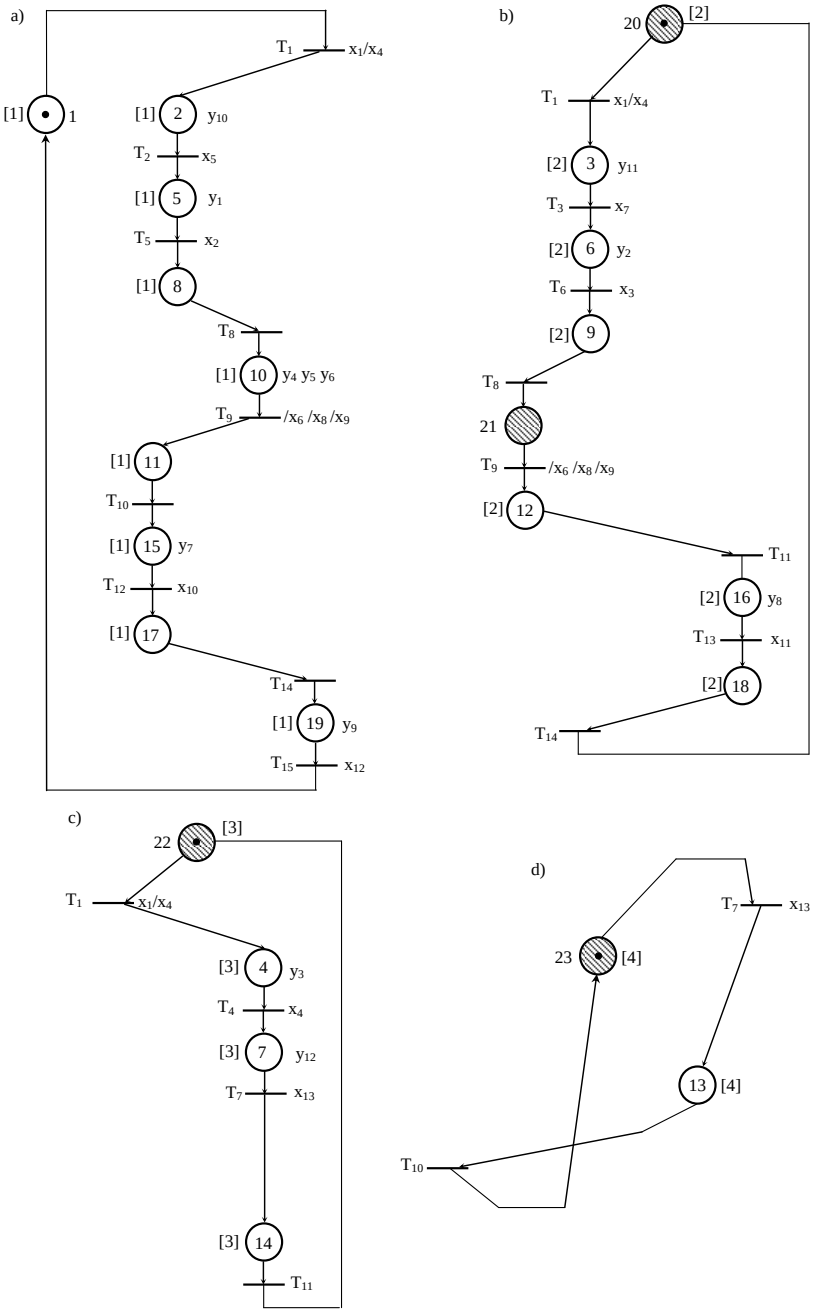

Fig. 6.9. Decomposed Petri net PN_2

The subsequent methods of realization were based on the application of the decomposition of a Petri net. Thus, the second model is a description of a system with parallel encoding. Since the decomposition resulted in four automaton subnets, each of them is encoded independently. In the system *PN2_ParallelEncoding*, presented in Appendix C, 10 flip-flops were used to encode all the nets, and the shared places are encoded with the application of superposition encoding (e.g., place p_1 is a superposition of the codes of the first, second and third automata).

The last of the developed models is the realization of a Petri net in the form of a modular digital system. The obtained SMCs were modelled independently as sequential automata, according to the scheme presented in Fig. 2.9. Also in this case, the final result of the implementation reveals the application of 10 flip-flops. However, the most essential fact is the application of FPGA dedicated memory blocks, which store outputs of particular sequential automata. Such an approach enables further decomposition of the memory block with the use of the method described in the subsequent chapter, which may be particularly essential for *system-on-a-chip* or *system-on-a-programmable-chip*, where the size of blocks may be limited (Wiśniewska *et al.*, 2005; Wiśniewski, 2009).

The behavioral simulation, carried out with the use of *Active-HDL*, system, demonstrated the relevance of the developed models. The outputs of the last of the systems (realization of a controller as a modular digital system) are delayed by half of a clock cycle in relation to the remaining models, which results from the application of synchronous memory of FPGA system (in order to maintain the proper functionality of the whole system, the memory blocks are active at the low edge of the clock signal). Figure 6.10 presents the obtained results of the simulation. The results obtained for all three versions of the controller are presented in the book. The values of the input signals (denoted with x), local states (p) and outputs (y for the first model, z for the second, v for the third) have been listed as a vector of values written in a hexadecimal notation.

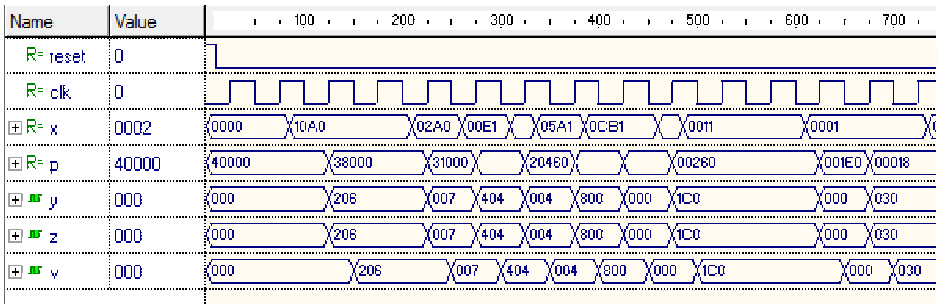

Fig. 6.10. Results of the simulation for net PN_2

6.2. Structural Decomposition of a Discrete System

The problem of microinstruction size reduction is an important issue in the process of designing digital systems. The algorithm is based on the determination and selection of compatibility classes of particular microoperations. Microoperations which are pairwise compatible may be encoded with the use of a smaller number of bits. The selection process of compatibility classes is NP-complete (Dasgupta, 1979; Robertson, 1979), therefore a number of methods which search for the solution were created (Hong *et al.*, 1990; Puri *et al.*, 1993). This subchapter presents an innovative method of compatibility class selection based on the theory of hypergraphs. The proposed solution has been thoroughly analyzed and compared to the traditional methods which are based on undirected graphs and operations on matrices.

6.2.1. Problem Formulation

Practically each major system exploits external memory to store information. For systems realized as system-on-a-chip (SoC), the capacity of the memory module closely relates to the production costs of the whole system. On the other hand, in complex solutions like system-on-a-programmable-chip (SoPC), in which FPGA dedicated memory blocks are used, the capacity of the memory is limited (Wilkes, 1951; Maxfield, 2004; Łuba, 2005). Therefore the reduction of the memory capacity is very frequently necessary. The operation may be performed in numerous ways, one of them is the decomposition, i.e., the division into smaller blocks. Another commonly used solution is the reduction of the memory word size through appropriate encoding of microinstructions (Wiśniewska, 2009; Wiśniewska *et al.*, 2009a). Thus, a total memory capacity may be reduced (Wiśniewska *et al.*, 2009b; Wiśniewska *et al.*, 2010). Obviously, the content of the memory is decoded at a later phase, but the process may be easily realized through simple logic blocks of the prototyped system, e.g., multiplexers (Wiśniewska *et al.*, 2005).

As mentioned before, the reduction algorithm of the microinstruction length is based on the selection of compatibility classes of particular microoperations. It is an NP-complete problem. A considerable number of algorithms improving the process selection of compatibility classes appeared in the world literature (Dasgupta, 1979; Hong *et al.*, 1990; Puri *et al.*, 1993; Malykh and Shakhnovskii, 2004; Wiśniewska and Wiśniewski, 2005). A vast majority of the solutions is based on stochastic and heuristic algorithms. Their undoubted advantage is their speed of operation since their computational complexity is essentially reduced. However, it should be clearly stated here that all the developed methods enable solving only one particular problem which is the reduction of the microinstruction size. This chapter presents a general method enabling the selection of compatibility classes with the use of the theory of hypergraphs. In practice it means that the solution can be obtained with **any known method for the determination of the smallest transversal of a hypergraph**.

The classical (standard) method is grounded on the application of the theory of undirected graphs. In the first phase, the relation mapping between particular

microoperations with the use of a compatibility graph takes place. If two microoperations are not performed at the same time, i.e., do not occur in the same microinstruction, they are connected with an edge. In the subsequent step, the class selection is performed. In practice it consists in the determination of the cliques in the compatibility graph. Next, the particular classes are described with the help of compatibility matrix (graph vertices are represented by columns, the edges by rows). The standard selection of compatibility classes relies on the matrix reduction. Its rows and columns are subsequently reduced (a detailed algorithm is described in the further part of the book). The remaining rows indicate the solution, i.e., the searched compatibility classes.

Although the term "hypergraph" has not been mentioned in the previous literature dealing with the reduction of microinstructions size, it is clearly visible that the compatibility matrix is nothing else but a matrix of *a compatibility hypergraph*. Compatibility classes correspond to its hyperedges, whereas microoperations included in these classes - to particular vertices. Analogically, the classical selection process is in fact the searched edge cover of a compatibility hypergraph (Wiśniewska *et al.*, 2011). It is similar to the fast reduction algorithm discussed in Chapter 4, proposed by (DeMicheli, 1994), which, in turn, is a modification of a method presented in (Rudell, 1989) (actually, it is the same algorithm, however introduces certain improvements and advances). Therefore this algorithm is recognized as a representative of classical solutions in the further considerations.

6.2.2. The Idea of the Proposed Method

The chapter presents an innovative method for the reduction of the microinstruction length. The algorithm is based on traditional solutions. Therefore two first steps of the presented method are identical with the standard methods. In the subsequent stages the proprietary solution using the theory of hypergraphs is applied.

The proposed method for the reduction of the microinstruction length may be divided into the following stages:

1. *The formation of set C_c of microoperation compatibility (coherence) classes.*

   Microoperations are compatible (coherent) if they are not performed concurrently, i.e., when they do not occur in the same microinstructions. The set of compatibility classes is represented as C_C=$\{C_1, \ldots, C_K\}$.

   The detailed description of how compatibility classes are determined may be found in publications recognized as classics in the field of the reduction of the microinstruction length (Hong *et al.*, 1990; Puri *et al.*, 1993).

2. *The determination of weight (cost) for each compatibility class.*

   The weight (cost) of a class is a minimal number of bits which are necessary to represent microoperations comprised in the class. The weight of the class may be easily determined from the formulation:

$$L_i = \lceil log_2(|C_i| + 1) \rceil. \tag{6.1}$$

The excessive bit is necessary to determine the state in which no encoded microoperation is performed. In the presented formula C_i denotes the i-th compatibility class, whereas L_i - the value of the weight of class C_i.

3. *The formation of the incidence matrix of a hypergraph for fixed compatibility classes.*

   The subsequent step is the formation of the incidence matrix of hypergraph H, which illustrates relations between microoperations. The vertices of the hypergraph correspond to the particular microoperations, whereas compatibility classes - to hyperedges. Formally, the incidence matrix of a hypergraph may take the value:

$$A_{ij} = \begin{cases} 1 & \rightarrow \text{ compatibility} \\ 0 & \rightarrow \text{ incompatibility} \end{cases},$$

   where $i = \{1, \ldots, K\}$ denotes the i-th compatibility class, $j = \{1, \ldots, N\}$ denotes the j-th microoperation. If entry A_{ij} of the incidence matrix contains value 1, it means that the j-th microoperation belongs to the i-th compatibility class.

4. *The formation of hypergraph H^* dual to hypergraph H.*

   In this step, a dual hypergraph is formed. In practice, the determination of a dual hypergraph consists in the transposition of incidence matrix A into transposed matrix A^*.

5. *The determination of the smallest transversal $\tau(H^*)$ of dual hypergraph H^*.*

   This stage may be realized with the use of any method enabling the calculation of the vertex cover of a hypergraph. The thesis involves the examination and comparison of all the methods for determining hypergraph transversals, which are described in Subchapter 4.2.2.

   It is worth mentioning that there is a possibility of obtaining more than a single smallest transversal. Therefore it is necessary to determine the best solution through the calculation of total cost, i.e., the number of bits necessary to encode the microoperations.

6. *The calculation of total cost for all the smallest transversals.*

   For each smallest transversal $\tau_S \in \tau$ the total cost (weight) W_s is calculated. The value may be determined from the formulation:

$$W_s = \sum_{i=1}^{I} L_i, \tag{6.2}$$

   where W_s denotes the total weight (total cost) of transversal τ_s. It is equal to the sum of weights L_i of all compatibility classes (i.e., it equals I), which belong to a given transversal. For further calculations, transversal τ_{Min} of the smallest total cost W_s is selected.

7. *Encoding compatibility classes which realize a minimal hypergraph transversal.*

   The subsequent step is to encode compatibility classes. The number of variables Q necessary for the realization of the task results directly from the weights assigned to these classes. The choice of the code is arbitrary, but the assignment of the first code (consisting of 0 exclusively) is recommended for these rows in which no microoperation is generated.

8. *The determination of a new content of memory with encoded compatibility classes.*

   The content of the memory is determined through juxtaposition of all variables obtained in the previous stages. It implies that the size of the memory word (microinstruction) after the reduction equals the sum of weights of all the classes used for the smallest cover:

$$t = \left(1 - \frac{\sum_{i=1}^{I} L_i}{N} \right) * 100\%, \tag{6.3}$$

   where:
   t - reduction (expressed in %) of the memory capacity;
   I - number of compatibility classes realizing the smallest vertex cover;
   L_i - weight of the i-th class comprised in the minimal cover;
   N - primary size of a microinstruction.

The system prepared in such a way may be easily realized with the use of digital systems. For example, in the case of FPGA programmable matrices, a decomposed memory may be implemented with the use of dedicated memory blocks, whereas the decoding particular microoperations is performed trough the logic blocks of an FPGA (e.g., LUTs).

6.2.3. An Example of Structural Decomposition

Let us consider memory M_1 (Tab. 6.1) containing N=6 outputs (microoperations) performed in M=4 microinstructions assigned to the same automaton subnet, which has been formed through the decomposition of an exemplary Petri net. It implies that the total initial capacity of the memory equals V_{pocz}=N*M=24 bits.

According to the proposed reduction algorithm of the microinstruction length, the first stage relies on the determination of compatibility classes. For the presented example, the set consists of K=4 elements: C_C=$\{C_1, \ldots, C_4\}$, where C_1=$\{y_1, y_2, y_3\}$, C_2=$\{y_1, y_3, y_4, y_6\}$, C_3=$\{y_2, y_5\}$ and C_4=$\{y_4, y_5, y_6\}$. The particular classes comprise sets of compatible elements, i.e., microoperations which

Tab. 6.1. Exemplary memory M_1

μ_M	Microoperation						M
	y_1	y_2	y_3	y_4	y_5	y_6	
μ_1	0	1	0	0	0	1	1
μ_2	0	1	0	1	0	0	2
μ_3	1	0	0	0	1	0	3
μ_4	0	0	1	0	1	0	4

do not occur in the same microinstructions. For example, set C_1 contains elements y_1, y_2 and y_3, which means that the three microoperations do not occur concurrently in any of the four microinstructions.

In the subsequent step, the weights of compatibility classes are determined. According to Formulation 6.1, the particular classes take the following weights:

$C_1 = \lceil log_2\ (3\ +\ 1)\ \rceil = 2,$
$C_2 = \lceil log_2\ (4\ +\ 1)\ \rceil = 3,$
$C_3 = \lceil log_2\ (2\ +\ 1)\ \rceil = 2,$
$C_4 = \lceil log_2\ (2\ +\ 1)\ \rceil = 2.$

The third stage of the reduction algorithm is the determination of the incidence matrix of the compatibility hypergraph. The hypergraph illustrates the relations between microoperations and compatibility classes. The graphical form of compatibility hypergraph H_{M_1} is presented in Fig. 6.11.

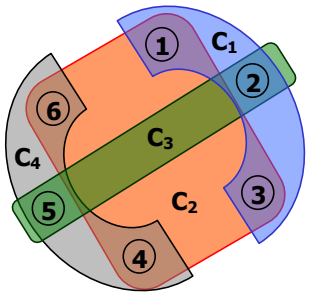

Fig. 6.11. Hypergraph H_{M_1}

For the considered example, compatibility hypergraph H_{M_1} consists of six vertices (corresponding to particular microoperations) and four hyperedges (representing compatibility classes). Since incidence matrix A_{M_1} consists of 24 fields, the rows correspond to the compatibility classes, whereas the columns - to the

microoperations. Figure 6.12 presents incidence matrix A_{M_1} of a compatibility hypergraph.

$$A_{M_1} = \begin{array}{c} \begin{matrix} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \end{matrix} \\ \left[\begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{matrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{matrix}$$

Fig. 6.12. Incidence matrix A_{M_1} of compatibility hypergraph H_{M_1}

Then a hypergraph dual to the compatibility hypergraph is determined. The edges of the dual hypergraph correspond to the vertices of the compatibility hypergraph, whereas its vertices - to the edges. The incidence matrix of dual hypergraph $A^*_{M_1}$ (Fig. 6.13) is a transposition of the matrix of the compatibility hypergraph.

$$A^*_{M_1} = \begin{array}{c} \begin{matrix} C_1 & C_2 & C_3 & C_4 \end{matrix} \\ \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{matrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{matrix}$$

Fig. 6.13. Incidence matrix of a dual hypergraph $A^*_{M_1}$

In the subsequent step, all smallest transversals of the dual hypergraph are determined. The operation may be performed with the use of any method calculating the smallest vertex cover of a hypergraph. It is a very important stage of the reduction process of the microinstruction size since the degree of the reduction is closely connected with the result of the performed selection. The cover with the smallest sum of weights of compatibility classes constitutes, in this case, the criterion of effectiveness. Obviously, the best solution is returned if exact algorithms are applied (e.g., back tracking), but it involves the exponential computational complexity (Robertson, 1979). Therefore, approximate solutions are frequently applied (a fast reduction algorithm, a greedy algorithm). A detailed description of the experiments and the examinations of the effectiveness of the discussed algorithms are given in Chapter 7.

For the considered dual hypergraph there exist two smallest transversals, both consisting of two elements: τ_1 realizes the cover with the use of edges C_1 and C_4, whereas τ_2 consists of C_2 and C_3. To select the best solution, it is necessary to

perform the subsequent stage of the reduction algorithm of the microinstruction.

For all determined smallest transversals, their total weight (cost) is calculated. According to the results obtained in the second stage of the proposed method, the weights of the particular classes are, respectively: C_1=C_3=C_4=2 and C_2=3. Thus, the total cost of the smallest transversals equals:

- for transversal τ_1: W_1=L_1+L_4=2+2=4;
- for transversal τ_2: W_2=L_2+L_3=3+2=5.

It follows that the cover realized by edges C_1 and C_4, should be selected for the further analysis, since the number of bits necessary for encoding compatibility classes is the smallest. The total number of bits clearly defines the number of variables which are used to encode microoperations. In the considered example $|Q|$=4, thus Q=$\{q_1, q_2, q_3, q_4\}$. Since both classes have the same weight, equal 2, therefore variables q_1 and q_2 are used to encode class C_1, whereas q_3 and q_4 represent microoperations comprised in C_4. The code selection is arbitrary. In the presented example, the natural binary code is used. Microoperation encoding is presented in Tab. 6.2.

Tab. 6.2. Compatibility class encoding

Class C_1			Code		Class C_4			Code	
y_1	y_2	y_3	q_1	q_2	y_4	y_5	y_6	q_3	q_4
0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	1	0	0	0	1
1	0	0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0	1	0

The final content of the memory with encoded microinstructions is determined on the basis of Table 6.2. Since $|Q|$=4 variables have been used in the encoding process, the new size of the microinstruction is four bits. Table 6.3 illustrates the content of the memory with the encoded microinstructions.

Tab. 6.3. Content of memory M_1 after the reduction of the microinstruction length

μ_M	Microoperation				M
	q_1	q_2	q_3	q_4	
μ_1	0	0	0	0	1
μ_2	0	0	0	1	2
μ_3	0	1	1	0	3
μ_4	1	0	1	0	4

The above table implies that the size of the encoded microinstruction is equal to four bites, thus the total capacity of the new memory equals $V_{końc}$=16 bits. Thus, it can be concluded that the initial memory V_{pocz} has been reduced by t=33%.

The illustrated example focuses on the demonstration the essence of the algorithm. However, the method for the reduction of a microinstruction has been thoroughly examined with the use of four different methods for the determination of the smallest transversal (a fast reduction algorithm, a backtracking algorithm, a greedy algorithm and a mixed algorithm). The detailed description and the results of the performed experiments are presented in Chapter 7.

The discussed example was modelled with the use of the Verilog language, and then implemented in a real programmable system FPGA (the system *XC3S500E* of *Spartan3E* family made by *Xilinx* was used) (Węgrzyn *et al.*, 1996; Zwolinski, 2000). The source code of both memory versions (before and after decomposition) is presented in Appendix C. The encoding operation was realized through the determination of logic formulas for the particular microoperations (module *Pamiec_zdekomponowana*).

The simulation confirmed the functional correctness of the performed reduction of the microinstruction length. The obtained results were identical for both the initial memory (microoperations denoted as y) and the decomposed memory (denoted as (z) (Fig. 6.14).

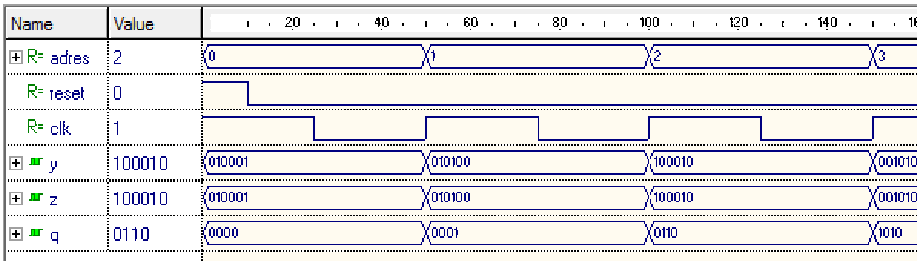

Fig. 6.14. Results of the simulation of memory M_1 before (y) and after (z) decomposition

6.3. Summary

The chapter presents two innovative methods for the decomposition of discrete systems. The first method refers to the parallel decomposition of concurrent automata described by Petri nets. The algorithm is based on the determination and selection of exact transversals in a concurrency hypergraph. The method searches for the subsequent automaton subnets in a polynomial time, which means that the computational complexity has been reduced in comparison to the methods used so

far (exponential complexity). The detailed results of the research have confirmed high efficiency of the proposed method, which proved to be nearly *100 times faster* than in average the algorithms operating on graphs.

The second proposed solution refers to the reduction of the memory capacity. The algorithm operates on the selection of the compatibility classes of microoperations on the basis of hypergraph transversals. An innovative solution, based on the application of an arbitrary method searching for the vertex cover in the compatibility hypergraph, has been proposed. The results of the research (Chapter 7) have revealed that the application of the greedy algorithm is the most effective. The method enables obtaining better results than the traditional method (on average, the memory is smaller by 16%) in a considerably shorter time (on average, 12 times faster).

An in-depth study was conducted for both methods. The detailed description of the experiments, the obtained results and their analysis are presented in Chapter 7.

Chapter 7

EXPERIMENTAL VERIFICATION OF DEVELOPED DECOMPOSITION METHODS

The decomposition methods for discrete systems presented in this work have been thoroughly examined. According to the description presented in Chapter 6, the conducted experiments were divided into two independent parts. The first part involved the examination of the effectiveness of the parallel decomposition of discrete systems, i.e., the decomposition of a Petri net into automaton subnets. The second part of the research included the structural decomposition of a memory module.

This chapter describes the details of the performed research: the author's tool supporting the research process, the applied libraries of testing modules, as well as the analysis of the obtained results of the experiments.

7.1. The Author's *Hippo* System Supporting the Process of Decomposition, Analysis and Design of Discrete Systems

To perform the verification of the experimental methods proposed in the book, the author developed the *Hippo*, system to support the decomposition of discrete systems. The author's direct motivation was the lack of such programs both in the world of computer science and mathematics. Although there are the programs and modules for information systems (such as, e.g., *Matlab*), which enable basic mathematical operations on graphs (cover, complement, coloring), they are strongly restricted and usually apply just one method of the result determination (most frequently it is a kind of greedy algorithm). Whereas, no professional tool performing (even basic) operations on hypergraphs had been encountered by the author. Therefore, it was decided to develop original tool to support the decomposition of discrete systems with the use of hypergraphs (Wiśniewska, 2011).

In most cases the *Hippo* system allows the application of several alternative algorithms in solving a given problem (e.g., 5 different algorithms of the determination of hypergraph transversals were implemented). Such an approach made it possible to perform complex experiments connected with the decomposition of discrete systems, and, in particular, the comparison of the effectiveness of graph and hypergraph methods. It is worth mentioning that the *Hippo* system finds a practical application in the Institute of Computer Engineering and Electronics at the University of Zielona Góra. What is more, it is used in the realizations

of major projects presented, among others, during the International Science Picnic in Warsaw or the Festivals of Science in Zielona Góra. Therefore, the initial version of the system has been significantly expanded by a series of o subsequent modules. At present the *Hippo* system is a complex tool that supports the process of decomposition, analysis and design of discrete systems.

This subchapter focuses on the application of the system for the verification of the effectiveness of the developed methods. The detailed description of all modules is given in Appendix A. In order to conduct the research connected with the decomposition of discrete systems the following modules were developed:

- ⋆ *Transversals* - a module enabling the determination of the vertex cover in a hypergraph, 5 different algorithms (fast, greedy, backtracking, and mixed algorithms as well as *DLX* for the determination of exact transversals) were implemented. The module was used both during the experiments connected with the parallel decomposition (exact transversals in the concurrency hypergraph) and in the structural decomposition (the determination of the smallest transversal in the selection of compatibility classes).

- ⋆ *Coloring* - a module enabling graph or hypergraph coloring, 6 different coloring methods were implemented, including 4 finding an approximate solution (unordered coloring, SL-ordered coloring, LF-ordered and randomly ordered coloring) and 2 finding an exact solution (backtracking algorithm and for exact hypergraphs - coloring through the determination and selection of exact transversals). The module formed the research basis for the parallel decomposition (coloring of a graph or concurrency hypergraph vertices).

- ⋆ *Dualism* - a module finding a hypergraph dual to a given hypergraph, constitutes an integral part of the two decomposition processes (in parallel decomposition during the determination of the selection hypergraph and in structural decomposition during the selection of compatibility classes).

- ⋆ *Complement* - a module enabling the determination of a complement for a given graph or hypergraph. Two methods were implemented, and both of them find an exact solution (a backtracking algorithm, and in the case of c-exact hypergraphs - the complement is found through the determination of all exact transversals). The module was applied in the formation process of compatibility classes, which enabled the verification of the developed method for the structural decomposition with the use of real memory modules.

The developed modules constituted the basis for the performed experiments. During the experiments, the efficiency of the developed methods was tested, by comparing the performance times and the obtained results. In order to achieve the most reliable results, the proposed solutions were compared with the representatives of the algorithms used currently, (for the parallel decomposition it was graph coloring, whereas for structural decomposition - the selection of the compatibility classes with the use of fast reduction algorithm).

7.2. The Research Connected with Parallel Decomposition

7.2.1. The Library of Test Modules and Research Methodology

The developed method for parallel decomposition of a Petri net with the application of determination and selection of exact transversals in the concurrency hypergraph was compared to the traditional solutions based on coloring of a classical concurrency graph. For this purpose, the *benchmarks* elaborated at the University of Zielona Góra were used. The examined examples were based on both - Petri nets presenting real discrete systems (such as the controller of a volumetric feeder, a system of a chemical reactor, etc.) and hypothetical systems. What is more, all the systems described above were correctly formed Petri nets. The main objective of the performed experiments were the verification of the efficiency of the proposed methods for the decomposition of discrete systems, based on the application of exact transversals. The efficiency criterion was determined by the time of the algorithm performance in relation to the traditional method based on concurrency graph coloring. The obtained result (the number of the achieved automaton subnets) should be identical in both cases, as both methods are assumed to find the optimal solutions, i.e., the smallest possible number of SMCs into which the original Petri net can be decomposed.

7.2.2. The Research Results

Table 7.1 presents the averaged results of the research obtained in the conducted experiments. The table lists the results of the decomposition of Petri nets with the use of three methods for the decomposition of Petri nets. First two ways were based on graph and concurrency hypergraph coloring with the use of the exact method (a backtracking algorithm). The third method was based on the determination and selection of exact transversals in a concurrency hypergraph. The table includes respectively: the average number of vertices (number of the Petri net places), the average number of edges in the given graph or concurrency hypergraph, the average number of the obtained concurrent subnets (SMCs) and the average execution time of the algorithm.

The present research results presented above clearly show the superiority of the proposed method over the solutions used so far. **The realization of the algorithm, based on the determination and the selection of the exact transversals in the concurrency hypergraph, was nearly one hundred times faster in average than in the previously used solutions, and the obtained (expected) results were identical for all the cases** (detailed results are presented in Appendix B).

The analysis of the obtained results for the particular test nets leads to the conclusion that the efficiency of the algorithm depends strongly on the target number of SMCs into which the original Petri net is decomposed. If a Petri net is divided into a maximum of two SMCs, the proposed method may appear to be considerably slower than classical solutions. However, it should be clearly stressed here that this case is unique and occurs rather seldom. Yet in the case of three

Tab. 7.1. Averaged results of the research of the parallel decomposition of discrete systems

	Concurrency graph coloring	Concurrency hyper-graph coloring	Calculation and selection of exact transversals
Average number of vertices	11,44	11,44	11,44
Average number of edges	35,06	16,04	16,04
Average number of SMCs	3,28	3,28	3,28
Average execution time of the algorithm [ms]	4441,06	4325,45	**45,58**

SMCs, the algorithm based on the calculation and selection of exact transversals occurs to be distinctly faster than the methods based on the graph or concurrency hypergraph coloring. **The efficiency of the proposed method increases with the number of the target SMCs**, which is clearly visible in Fig. 7.1 (the averaged results of all the examined test modules).

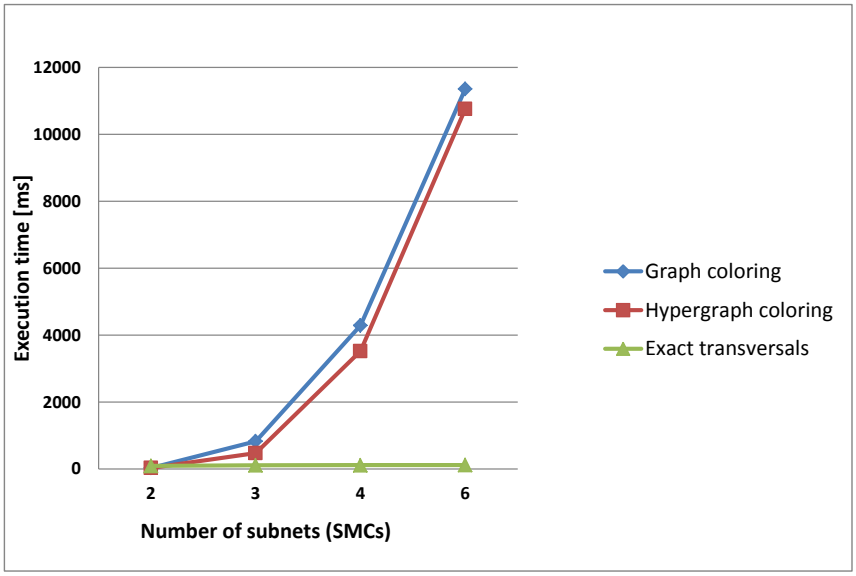

Fig. 7.1. Execution time for algorithms in relation to the number of SMCs

Among the real discrete systems (the philosophers' problem, beverage production control), in which the number of automaton subnets ranged from 4 - 6, the time of the solution determination with the use of exact transversals was over 100 times shorter than for the method applying concurrency graph coloring.

7.3. The Research Connected with the Structural Decomposition

7.3.1. The Library of Test Modules and Research Methodology

The second important aspect, of the research and experiments was the determination of the efficiency of the proposed method for the reduction of the microinstruction length. The method was verified experimentally with the use of four different algorithms, realizing the hypergraph vertex cover. The fast reduction algorithm (a representative of a classical method, most frequently used nowadays), greedy algorithm, backtracking algorithm and hybrid algorithm (a mixture of fast reduction algorithm and the greedy algorithm) were examined. All the methods were verified with the use of over 100 test memories. The vast majority of the examined tests (about 70) were real memory systems of logic controllers, taken from the libraries provided by S. Baranov (Holon Institute of Technology) and the test base of the University of Zielona Góra (teams led by L. Titarenko, A. Barkalov and M. Adamski). The remaining modules are hypothetical memories (randomly generated).

Each of the test memories was subjected to a full path of the reduction of the microinstruction length. Thus, at the beginning, compatibility classes were determined, and the weights for these classes were established. Then on this basis, hypergraph H, illustrating the relations between the particular microoperations, was formed, and next - according to the algorithm described in Chapter 6 - dual hypergraph H^*. In the subsequent step, the smallest transversals for hypergraph H^* were determined. This stage was the main objective of the performed experiments. Four different methods for the determination of transversals were examined and compared. The execution time and the size of the found transversal were assumed to be the comparative criteria (the costs of particular weights were taken into account, according to the algorithm presented in Chapter 6).

7.3.2. The Research Results

Table 7.2 presents the averaged results of the conducted research. The columns present the results obtained during the determination of the hypergraph transversals with the use of the particular algorithms. The rows illustrate the efficiency of a given algorithm - the proportional reduction of the total capacity of a memory and the averaged time of the execution of a given algorithm.
Table 7.3 presents the results of the research in relation to the fast reduction algorithm in order to illustrate and compare the algorithms better. As mentioned in Chapter 6, the method was selected as a representation of traditional solutions based on matrix operations (Dasgupta, 1979; Robertson, 1979).

Tab. 7.2. Reduction of the microinstruction length - averaged research results

	Fast reduction algorithm	Greedy algorithm	Backtracking algorithm	Hybrid algorithm
Reduction of the capacity of a memory [%]	82%	69%	**66%**	74%
Average time of the execution [s]	0,31273	**0,02625**	$40,8067^{[1]}$	0,37214

[1] - due to the exponential computational complexity of the algorithm, a part of the tests was stopped after an hour.

Tab. 7.3. Reduction of the microinstruction length - a comparison of algorithms

	Greedy algorithm	Backtracking algorithm	Hybrid algorithm
Reduction of the capacity in relation to the fast reduction algorithm	84%	**81%**	92%
Reduction of the time in relation to the fast reduction algorithm	**8%**	$13048\%^{[1]}$	119%

[1] - due to the exponential computational complexity of the algorithm, a part of the tests was stopped after an hour.

According to the expectations, the backtracking algorithm appeared to be the most efficient (on average by 29% better than the traditional solutions based on the fast reduction algorithm). The fundamental disadvantage of the method is its exponential computational complexity, which in practice precludes it from being used in relatively large memories (it makes sense to apply the method for the memory sizes up to about 400 bits, i.e., 20 microinstructions x 20 microoperations). Surprisingly good effects were obtained for the greedy algorithm. On average, the method was by as much as 16% better than the traditional solution (the fast reduction algorithm), and was only slightly exceeded by the backtracking algorithm. Combining it with the definitely shortest execution time, this solution appeared to be the most efficient for larger memory modules.

7.4. Summary

The chapter presents a means of experimental verification used for the developed method for decomposition of discrete systems. The base for the conducted research was the author's *Hippo* system, in which graph and hypergraph algorithms were implemented. The obtained empirical results confirm the high efficiency of the proposed method for the parallel decomposition of concurrent automata. The application of the exact transversals accelerates the process of solution finding

by nearly one hundred times, in comparison to the solutions based on classical concurrency graph coloring. While in the case of structural decomposition, the efficiency of algorithms based on the determination of the smallest transversal in a compatibility hypergraph was tested. The research demonstrates surprisingly good results for approximate solutions (a greedy algorithm), which are on average 16% more efficient (smaller memory capacity resulting from the decomposition) and 12 times faster than the generally used methods.

Chapter 8

SUMMARY

Recent years have observed a dynamic development in electronic and information technologies. It is connected with the growth of the realized discrete systems, followed by the continuous improvement of design methods and tools. The main objective of the developed algorithms is to obtain satisfactory results in the shortest possible time. The vast majority of problems are currently realized with the use of the classical theory of undirected graphs, which a few years fully complied with the set objectives. However, the increase in the size of discrete systems necessitates the continuous modifications of the existing methods for decomposition, analysis or design, and the search for the new ones. The introduction of the theory of hypergraphs may appear to be a kind of a breakthrough. Despite the fact that hypergraphs were proposed already in the 70s of the previous century, it is only in recent years that they have been more and more boldly used in the issues connected with mathematics, computer science or electronics. Furthermore, very interesting properties of these structures cause the rapid growth of interest of the researchers from all over the world, which in turn results in numerous practical applications of hypergraphs in problem-solving in various domains, such as mathematics (Elbassioni and Rauf, 2010), chemistry (Konstantinova and Skorobogatov, 2001), and first of all computer science and electronics (Eiter and Gottlob, 2002).

In the book two methods for decomposition of discrete systems are proposed. Both solutions are based on the application of the theory of hypergraphs. The first of the methods is connected with the parallel decomposition of concurrent automata, described by Petri nets. The controller is divided into the sequential automata cooperating with each other, whereas each of them may be designed independently, e.g., as a controlling system with a memory. The second of the proposed solutions refers to the structural decomposition of a discrete system. In the process of decomposition, the capacity of the memory block is reduced (particularly for memory blocks of sequential automata) by applying the reduction of the microinstruction length.

The main objective of the developed methods is to improve the decomposition process through the reduction of the algorithm execution time (both for the parallel and structural decompositions), as well as to find a more effective solution (for structural decomposition, the effectiveness is understood as obtaining a smaller size of the decomposed memory), in comparison to the currently used solutions.

This chapter constitutes a synthetic summary of the research and development connected with the decomposition of discrete systems. It presents the innovative elements of the work developed by the author herself. The chapter comments on the thesis as well as the objectives of the work. Moreover, it outlines possible directions of further research.

8.1. Confirmation of the Thesis

The thesis of the work has been theoretically justified and experimentally confirmed. The book shows that the application of hypergraphs in both parallel and structural decompositions considerably improves the processes in comparison to the generally used solutions.

Theoretical considerations, presented in Chapters 4, 5 and 6, connected with the parallel decomposition of concurrent automata described by Petri nets, show that the determination of the subsequent exact transversals in a concurrency hypergraph may be performed in polynomial time. It means that the computational complexity was considerably reduced in comparison to the commonly used solutions in which finding the subsequent SMCs is connected with the exponential complexity.

The experiments confirmed the high efficiency of the developed method for the decomposition of discrete systems. **The execution time of the algorithm based on the determination and selection of exact transversals in a concurrency hypergraph was nearly one hundred times faster in average than the solutions used so far, and identical results (expected) were obtained in all the cases** (the detailed results are presented in Appendix B).

The analysis of the obtained results for the particular test nets allows the statement that the algorithm effectiveness strongly depends on the target number of SMCs into which the original Petri net was decomposed. If a Petri net is divided into a maximum of two subnets, the proposed method may turn out to be even slower than the classical solutions. However, it should be stressed clearly that it is a special and rare case. Already in the case of three subnets, the algorithm based on the calculation and selection of exact transversals turns out to be distinctly faster than the methods based on graph or concurrency hypergraph coloring. **The effectiveness of the proposed method increases with the number of target SMCs**. Among the real discrete systems (philosophers problem, beverage production control) in which the number of subnets ranged from 4 to 6, the time of finding the solution with the use of exact transversals was more than 100 times shorter than for concurrency graph coloring.

The application of a hypergraph in the selection of compatibility classes in the reduction of the microinstruction length allows not only reducing the time necessary for obtaining the results, but it also significantly improves the reduction algorithm. The performed experiments showed that the greedy algorithm is on average 12 times faster and by 16% more effective than the commonly used method which is based on the fast reduction algorithm (the effectiveness directly causes the smaller memory capacity after the decomposition). It is worth mentioning that for

the selection of compatibility classes **any arbitrary algorithm determining the smallest hypergraph transversal** may be used, which opens future possibilities to apply new, today unknown methods of finding the hypergraph vertex cover. The most essential results of the research have been presented at conferences and published in international and national journals.

The decomposition algorithms developed in the work and the *Hippo* system have found practical application in the Institute of Computer Engineering and Electronics at the University of Zielona Góra (both in teaching and in scientific research). The program was used in the realization of the projects presented during the International Science Picnic in Warsaw (2011), Festivals of Science in Zielona Góra (2010 and 2011), as well as during the Days of Lubuskie Province (2010).

This book was realized within the research project financed from the Integrated Regional Development Operational Programme funds and co-financed by the European Social Fund (both projects in the years 2005-2006) and Sub-measure 8.2.2 "Regional Innovation Strategies", Measure 8.2 "Transfer of knowledge", Priority VIII "Regional Human Resources" for the economy of the "Human Capital Operational Program" co-financed by the European Social Fund and the State budget (2010-2011).

8.2. Innovative and Author's Elements

The confirmation of the main thesis resulted in the innovative methods and ideas developed by the author. The most essential and innovative elements introduced in the work are as follows:

- ⋆ the development of a method for parallel decomposition of discrete systems described by Petri nets with the application of the theory of hypergraphs whose aim was to improve the decomposition process of digital concurrent automata;
- ⋆ the development of a method for structural decomposition of discrete systems based on the reduction of the memory microinstruction length (in particular, the memory of sequential automata of the decomposed concurrent automaton) with the use of the theory of hypergraphs;
- ⋆ the introduction of new definitions connected with the application of the theory of hypergraphs into the decomposition of discrete systems (terms *the concurrency hypergraph*, *the sequentiality hypergraph* and *the selection hypergraph*);

- ⋆ the development of theorems connected with the application of the theory of hypergraphs into the decomposition of discrete systems (it was proved, among others, that the first and the subsequent exact transversals in a concurrency hypergraph may by determined in polynomial time).
- ⋆ the development of the author's *Hippo* system supporting the design process, analysis and decomposition of discrete systems with the use of both hypergraphs and traditional undirected graphs;
- ⋆ the experimental verification of the effectiveness of the developed methods for discrete system decomposition.

8.3. Directions of Further Work

The developed methods for decomposition of discrete systems may constitute the base for further research. One of the possible directions of the work is to particularize the number and size of the obtained SMCs in parallel decomposition. Currently, the algorithm returns the smallest possible number of subnets which, for example, from the point of view of a digital system designer is not always beneficial. It may sometimes happen that the effectiveness criterion will be constituted by the smallest and possibly regular number of states in all decomposed SMCs, with little attention to their general number. It is also possible to particularize the search criterion (the exact number of expected subnets, their minimal/maximal size, etc.). Obviously, the realization of the tasks formulated in such a way will be connected with the extension of the decomposition process. However, the effectiveness of an algorithm based on the determination of subsequent exact transversals should increase in comparison to the traditional solutions (in which the solution may be even impossible to be found).

The book highlights the series of very interesting properties of exact hypergraphs. Therefore the structure is currently the subject of intense research, primarily because of the possibility to reduce the computational complexity of algorithms. So far, it has been proved that the subsequent exact transversals may be determined in polynomial time, but the exponential number of transversals still remains a problem. The work presents a method based on a partial search for the solution set until all vertices of the concurrency hypergraph are covered by exact transversals. Although no theoretical proof was carried out (which is due to advanced mathematical algorithms), the research and the experiments suggest that the whole process of determination and selection of exact transversals may be connected with polynomial computational complexity. It could result in the possibility of parallel decomposition of Petri nets in polynomial time. The research into this field may provide a very interesting work, particularly due to the fact that exact transversals present a wide spectrum of applications, from mathematics and geometry (Elbassioni and Rauf, 2010), through entertainment, e.g., finding solutions to *Sudoku* puzzles or problem of placing queens on a chessboard (Knuth, 2000), to computer science and discrete systems (Eiter, 1994; Wiśniewska *et al.*, 2007b).

Appendix A

HIPPO - AUTHOR'S SYSTEM SUPPORTING THE PROCESS OF DECOMPOSITION OF DISCRETE SYSTEMS

In order to improve the process of decomposition of discrete systems and automation of the performed research and experiments, the author's *Hippo* system was developed. The tool consists of the set of three modules, realizing the most essential operations concerning the theory of graphs and hypergraphs (among others coloring, covering, complement, dualism, etc.). Figure A.1 demonstrates a graphical representation of the *Hippo* system.

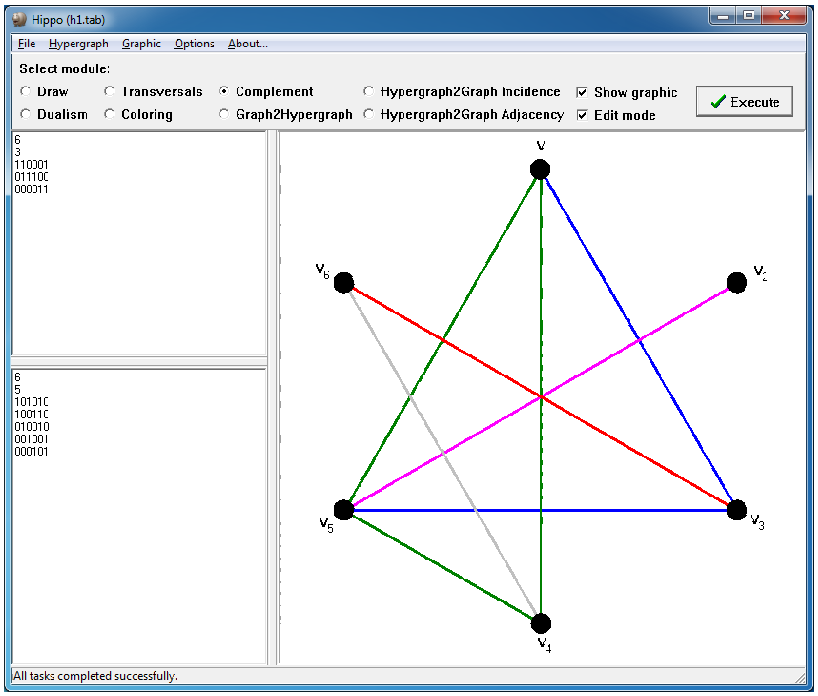

Fig. A.1. Graphical representation of the *Hippo* system

Despite a very detailed analysis and the thorough searches on the Internet, the author did not find any systems supporting the calculation process which would exploit the theory of hypergraphs. Although there are ready-made applications (usually commercial ones) supporting the graph operation processes, the programmes operating on the theory of hypergraphs were not encountered. Therefore the author decided to realize her own system which was intended for the verification and automation of the developed methods for the decomposition presented in Chapter 6. Eventually, the project was significantly expanded and a series of additional algorithms and methods (e.g., conversions between various formats, graphical representation, the possibility of exporting vector graphics, etc.) were implemented. In this case, the author was motivated by the practical use of the system in the Institute of Computer Engineering and Electronics at the University of Zielona Góra (both in teaching, promotion and scientific research).

A.1. Input Data for the *Hippo* System

The Hippo system operates on matrix structures. Standard input consists of a description of a graph or hypergraph presented in the form of a text. Output data may be transferred as a text file or directly in a graphical environment. The structure of default input data is close to the incidence matrix of a graph/hypergraph:

- ⋆ The first row defines the number of vertices of a graph/hypergraph
- ⋆ The second row contains the number of edges of a graph/hypergraph
- ⋆ The subsequent rows contain the values of the incidence matrix of a graph/hypergraph. Value *1* means that the edge represented by the j-th column contains a vertex represented by the i-th row. Analogically, value *0* is understood as the lack of a vertex belonging to a given column.

Fig. A.2. Input data of the *Hippo* system

Fig. A.2 presents a hypothetical description of a hypergraph of m=6 vertices (V=$\{v_1, v_2, v_3, v_4, v_5, v_6\}$) and n=3 hyperedges (E=$\{E_1, E_2, E_3\}$). In the considered example the particular edges contain the following vertices: E_1=$\{v_1, v_2, v_6\}$, E_2=$\{v_2, v_3, v_4\}$, E_1=$\{v_5, v_6\}$.

Additionally, input data may be presented as an adjacency matrix, (obviously, it makes sense only in the case of graphs). It should be remembered, though, that the representation may only be used in the combination with *Graph2Hipergraph*, since the other modules are compatible only with the default data format.

A.2. The Results of the Operation Execution in the *Hippo* System

Depending on the used module, the system may return data in various formats. In the case of modules *Coloring* and *Transversals* the result is a text description of the obtained results (Fig. A.3). The remaining modules will return the result in the form of an incidence (or adjacency) matrix. The result is graphically illustrated which considerably enhances the readability of the obtained results. There is also a possibility of displaying all the potential results of coloring and transversals through the selection of an appropriate method (Fig. A.3).

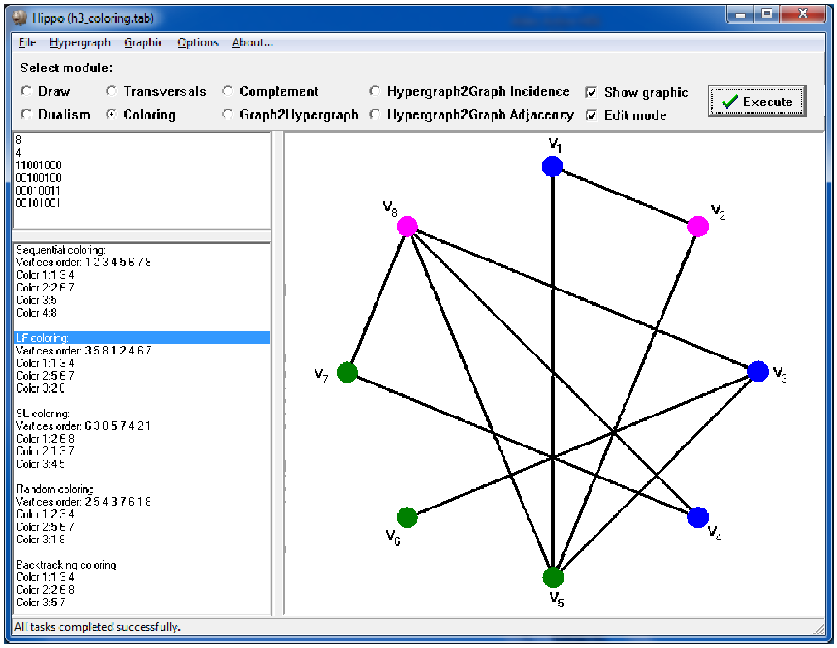

Fig. A.3. Results of the *Hippo* system

A.3. Structure of the *Hippo* System

The *Hippo* system contains eight independent modules, and each is performed as an independent process. The *Hippo* comprises the following modules:

1. *Complement* - a module determining the complement of a given graph/hypergraph.

2. *Coloring* - a module realizing the coloring of a graph/hypergraph. Five methods of graph and hypergraph coloring were implemented, four of which

are based on the greedy algorithm (unordered, LF-type ordered, SL-type ordered and randomly ordered). Moreover, there is a possibility of applying exact methods through the use of the backtracking algorithm (both for graphs and hypergraphs) and through the determination and selection of exact transversals (only for c-exact hypergraphs).

3. *Transversals* - a module calculating the cover. Four methods for the determination of transversals were implemented: the fast reduction algorithm, greedy algorithm, backtracking algorithm, hybrid algorithm (a combination of the fast reduction algorithm and the greedy method). Moreover, the algorithm calculating the exact transversals in the c-exact hypergraph was implemented. The method is based on a substantially modified D. E. Knuth's *DLX* algorithm, in which the possibility of interrupting the searching process was introduced. Such a possibility may occur only then when the smallest set of exact transversals which cover all the hypergraph vertices has been determined. The improved algorithm was applied in the parallel decomposition of discrete systems (Chapter 6).

4. *Dualism* - a module determining a hypergraph dual to a given hypergraph (transposition of the incidence matrix of a hypergraph).

5. *Graph2Hypergraph* - a module converting a graph into a hypergraph. In practice, the functionality of this module reduces to the determination of all maximal complete subgraphs, and then replacing them with their hyperedges. The input may be an adjacency matrix or an incidence matrix.

6. *Hypergraph2Graph Incidence* - a module converting a hypergraph into a graph (the conversion of a hypergraph incidence matrix into a graph incidence matrix).

7. *Hypergraph2Graph Adjacency* - a module converting a hypergraph into a graph (the conversion of a hypergraph incidence matrix into a graph adjacency matrix).

8. *Hypergraph2Tex* - man additional module converting a hypergraph (or a graph) described by an incidence (or adjacency) matrix into the form of a table interpreted by the LaTeXsystem.

Apart from these modules, in the *Hippo* system there are also the options facilitating and improving the operations on graphs and hypergraphs, e.g., there is a possibility to record the resultant picture in a vector form or turn off graphics mode (particularly useful in the case of relatively large structures where the graphical representation is illegible).

The basic objective of the developed modules was the automation of the decomposition of discrete systems. Therefore the most essential algorithms for graph and hypergraph coloring and covering were implemented (the investigation into the usability and efficiency of the developed methods for decomposition of discrete systems).

Appendix B

RESEARCH RESULTS

This appendix presents the detailed results of the research connected with the decomposition of discrete systems with the use of hypergraphs. The attached tables present the obtained results for the representative test modules, what is more the obtained results are briefly characterized.

B.1. The Research on Parallel Decomposition

Tables B.1 and B.2 present the results of the research associated with the parallel decomposition of discrete systems described by Petri nets. Sixteen test modules were selected for the list, some of which describe real discrete processes, whereas the others present a hypothetical and structurally correct Petri net (its name starts with the word *"Test"*).

Particular columns contain the following data:

- ⋆ *Benchmark* - specifies the name of the used test module;
- ⋆ *Algorithm* - a kind of applied decomposition algorithm (coloring of a concurrency graph, coloring of a concurrency hypergraph, determination and selection of exact transversals in a concurrency hypergraph);
- ⋆ *Number of vertices* - a number of concurrency graph/hypergraph vertices;
- ⋆ *Number of edges* - a number of concurrency graph/hypergraph edges;
- ⋆ *Number of SMCs* - a number of subnets (SMCs) obtained in the decomposition process;
- ⋆ *Algorithm run-time* - the time obtained in the decomposition with the use of a specific algorithm;
- ⋆ *Run-time comparison* - the ratio of the concurrency graph coloring run-time to the given decomposition algorithm run-time;
- ⋆ *Execution speed comparison* - the comparison of the given decomposition algorithm run-time to the concurrency graph coloring run-time.

Tab. B.1. The results of the research obtained for the parallel decomposition of discrete systems described by Petri nets (I)

Benchmark	Algorithm[1]	Number of vertices	Number of edges	Number of SMCs	Algorithm run-time [s]	Run-time comparison [%] (in relation to graphs)[2]	Execution speed comparison (in relation to graphs)[3]
IEC	graph	15	46	3	4,29807	-	-
	hypergraph	15	24	3	2,92737	68,11%	1,47
	transversals	15	24	3	0,198284	4,61%	21,68
Test04	graph	16	122	4	10,0333	-	-
	hypergraph	16	29	4	7,53286	75,08%	1,33
	transversals	16	29	4	0,224798	2,24%	44,63
Test07	graph	13	28	3	0,508774	-	-
	hypergraph	13	15	3	0,343154	67,45%	1,48
	transversals	13	15	3	0,176764	34,74%	2,88
Test11	graph	11	81	3	2,54041	-	-
	hypergraph	11	29	3	1,04675	41,20%	2,43
	transversals	11	29	3	0,0599462	2,36%	42,38
Test15	graph	13	74	3	0,373895	-	-
	hypergraph	13	26	3	0,203279	54,37%	1,84
	transversals	13	26	3	0,185218	49,54%	2,02
Test24	graph	11	11	2	0,0376585	-	-
	hypergraph	11	11	2	0,0376585	100,00%	1,00
	transversals	11	11	2	0,109133	289,80%	0,35
Test25	graph	13	64	3	3,07878	-	-
	hypergraph	13	32	3	1,82836	59,39%	1,68
	transversals	13	32	3	0,0680159	2,21%	45,27

[1] - subsequent rows determine the following methods: graph coloring, hypergraph coloring, determination of exact transversals.

[2] - the ratio of run-time with the application of the graph coloring method to the run-time with the use of a given algorithm.

[3] - the ratio of run-time with the use of a given algorithm to the run-time with the application of the graph coloring method.

Tab. B.2. The results of the research obtained in the parallel decomposition of discrete systems described by Petri nets (II)

Benchmark	Algorithm[1]	Number of vertices	Number of edges	Number of SMCs	Algorithm run-time [s]	Run-time comparison [%] (in relation to graphs)[2]	Execution speed comparison (in relation to graphs)[3]
Test31	graph	6	8	3	0,11067	-	-
	hypergraph	6	4	3	0,0545664	49,31%	2,03
	transversals	6	4	3	0,0530293	47,92%	2,09
Test33	graph	13	42	3	0,189445	-	-
	hypergraph	13	22	3	0,126809	66,94%	1,49
	transversals	13	22	3	0,185218	97,77%	1,02
Test35	graph	14	96	4	10,0222	-	-
	hypergraph	14	32	4	10,0114	99,89%	1,00
	transversals	14	32	4	0,0799283	0,80%	125,39
Reactor	graph	10	24	4	4,17779	-	-
	hypergraph	10	10	4	2,52658	60,48%	1,65
	transversals	10	10	4	0,102216	2,45%	40,87
Reactor 2	graph	10	24	4	2,88357	-	-
	hypergraph	10	10	4	1,64775	57,14%	1,75
	transversals	10	10	4	0,0599462	2,08%	48,10
Mixer	graph	11	29	4	2,32099	-	-
	hypergraph	11	10	4	1,07058	46,13%	2,17
	transversals	11	10	4	0,129115	5,56%	17,98
Philosophers	graph	14	143	6	10,1244	-	-
	hypergraph	14	22	6	10,0076	98,85%	1,01
	transversals	14	22	6	0,103369	1,02%	97,94

[1] - subsequent rows determine the following methods: graph coloring, hypergraph coloring, determination of exact transversals.

[2] - the ratio of run-time with the application of the graph coloring method to the run-time with the use of a given algorithm.

[3] - the ratio of run-time with the use of a given algorithm to the run-time with the application of the graph coloring method.

B.2. The Research on Structural Decomposition

Tables B.3 and B.4 present the results of the research connected with the decomposition of 10 selected benchmarks. The particular columns contain the following data:

- ⋆ *Benchmark* - specifies the name of the used benchmark (test module);
- ⋆ *Initial capacity* - the initial size of the test module memory;
- ⋆ *Algorithm* - a kind of the reduction algorithm;
- ⋆ *Number of classes* - a number of compatibility classes obtained as a result of the reduction;
- ⋆ *Number of variables* - a number of variables used to encode compatibility classes;
- ⋆ *Run-time* - the time obtained during the reduction with a given algorithm;
- ⋆ *Capacity after the reduction* - the capacity of memory after the reduction with the use of a given algorithm;
- ⋆ *Reduction of the memory capacity* - the capacity of memory after the reduction in relation to the initial capacity (expressed in percentage terms);
- ⋆ *Relative time reduction* - the time of reduction with the use of a given algorithm in relation to the time of reduction with the use of the classical algorithm (i.e., the fast reduction algorithm);
- ⋆ *Relative capacity reduction* - the capacity obtained in the reduction with the use of a given algorithm in relation to the capacity obtained with the use of the fast reduction algorithm.

The presented results demonstrate that only in a few instances the reduction with the use of the greedy algorithm gave worse results than the reduction with the use of the backtracking method. It is worth emphasizing that in the case of relatively large memories the reduction with the use of the exact algorithm was interrupted after one hour (the run-time for the greedy method was shorter than one second).

Tab. B.3. The results of the research obtained in the reduction of the microinstruction length (I)

Benchmark	Initial capacity [bits]	Algorithm	Number of classes	Number of variables	Run-time [s]	Capacity after the reduction [bits]	Reduction of the memory capacity[1]	Relative time reduction[2]	Relative capacity reduction[3]
mem_011	56	fast reduction	2	6	0,0095	42	75%	-	-
		greedy	2	6	0,00783	42	75%	82,35%	100%
		backtracking	2	6	0,23578	42	75%	2482,35%	100%
		hybrid	2	6	0,00922	42	75%	97,06%	100%
mem_037	24	fast reduction	4	9	0,01369	54	225%	-	-
		greedy	2	5	0,0067	30	125%	48,98%	56%
		backtracking	2	4	0,24137	24	100%	1763,26%	44%
		hybrid	2	4	0,01341	24	100%	97,96%	44%
mem_rnd47	100	fast reduction	2	6	0,01425	60	60%	-	-
		greedy	3	10	0,00838	100	100%	58,82%	167%
		backtracking	2	6	0,23858	60	60%	1674,51%	100%
		hybrid	2	6	0,01341	60	60%	94,12%	100%
mem_rnd63	90	fast reduction	4	14	0,0176	140	156%	-	-
		greedy	2	7	0,00894	70	78%	50,79%	50%
		backtracking	2	7	0,24919	70	78%	1415,88%	50%
		hybrid	2	7	0,01843	70	78%	104,76%	50%
mem_ab4	88	fast reduction	3	9	0,01676	90	100%	-	-
		greedy	3	9	0,00894	90	100%	53,33%	100%
		backtracking	3	9	0,25199	90	100%	1503,33%	100%
		hybrid	3	9	0,01872	90	100%	111,67%	100%

[1] - the ratio of primary memory to the memory reduced with the use of a given algorithm.

[2] - the ratio of a memory reduced by the fast reduction algorithm to the memory reduced by a given algorithm.

[3] - the ratio of the time of reduction with the use of the fast reduction algorithm to the time of reduction with the use of a given algorithm.

Tab. B.4. The results of the research obtained in the reduction of the microinstruction length (II)

Benchmark	Initial capacity [bits]	Algorithm	Number of classes	Number of variables	Run-time [s]	Capacity after the reduction [bits]	Reduction of the memory capacity[1]	Relative time reduction[2]	Relative capacity reduction[3]
mem_ab17	99	fast reduction	2	8	0,01006	80	80%	-	-
		greedy	2	8	0,00643	80	80%	63,89%	100%
		backtracking	2	8	0,23439	80	80%	2330,56%	100%
		hybrid	2	8	0,01062	80	80%	105,56%	100%
mem_ab25	112	fast reduction	2	5	0,03017	40	36%	-	-
		greedy	2	5	0,00922	40	36%	30,56%	100%
		backtracking	2	5	0,3358	40	36%	1112,96%	100%
		hybrid	2	5	0,01816	40	36%	60,19%	100%
mem_rw179	485	fast reduction	4	16	0,62606	320	36%	-	-
		greedy	2	8	0,03241	160	18%	5,18%	50%
		backtracking	2	8	28,85	160	18%	4608,21%	50%
		hybrid	4	16	0,45034	320	36%	71,93%	100%
mem_rw222	620	fast reduction	4	16	260,035	320	52%	-	-
		greedy	2	8	0,37435	160	26%	0,14%	50%
		backtracking	-	-	>1h	-	-	-	-
		hybrid	4	16	229,349	320	52%	88,20%	100%
mem_rw226	748	fast reduction	7	28	645,556	616	82%	-	-
		greedy	2	8	0,84676	176	18%	24%	28,57%
		backtracking	-	-	>1h	-	-	-	-
		hybrid	6	24	930,182	528	39%	71%	85,71%

[1] - the ratio of memory to the memory reduced with the use of a given algorithm.
[2] - the ratio of a memory reduced by the fast reduction algorithm to the memory reduced by a given algorithm.
[3] - the ratio of the time of reduction with the use of the fast reduction algorithm to the time of reduction with the use of a given algorithm.

Appendix C

AN EXAMPLE OF THE DISCRETE SYSTEM DESCRIPTION WITH THE APPLICATION OF VERILOG LANGUAGE

This appendix shows the description of all the models of discrete systems that shown in Chapter 6. The models of parallel decomposition of a discrete system are presented in (C.1), and the models of structural decomposition of a discrete system are presented in (C.2). All the systems were described with Verilog-HDL language.

C.1. Parallel Decomposition of a Discrete System

Lst. C.1. Description of *Mixer* (beverage machine), "one-hot" encoding

```
1  //Method 1: encoding "one-hot", as a standard Petri-net
2  //
3  module PN2_OneHot(y,x,clk,reset);
4          output [1:12] y;
5          input [1:13] x;
6          input reset,clk;
7
8          reg [1:19] p;
9          wire [1:15] t;
10
11         assign t[1]=x[1]&~x[4]&p[1];
12         assign t[2]=x[5]&p[2];
13         assign t[3]=x[7]&p[3];
14         assign t[4]=x[4]&p[4];
15         assign t[5]=x[2]&p[5];
16         assign t[6]=x[3]&p[6];
17         assign t[7]=x[13]&p[7];
18         assign t[8]=p[8]&p[9];
19         assign t[9]=~x[6]&~x[8]&~x[9]&p[10];
20         assign t[10]=p[11]&p[13];
21         assign t[11]=p[12]&p[14];
22         assign t[12]=x[10]&p[15];
23         assign t[13]=x[11]&p[16];
```

```
24          assign t[14]=p[17]&p[18];
25          assign t[15]=x[12]&p[19];
26
27
28          assign y[1]=p[5];
29          assign y[2]=p[6];
30          assign y[3]=p[4];
31          assign y[4]=p[10];
32          assign y[5]=p[10];
33          assign y[6]=p[10];
34          assign y[7]=p[15];
35          assign y[8]=p[16];
36          assign y[9]=p[19];
37          assign y[10]=p[2];
38          assign y[11]=p[3];
39          assign y[12]=p[7];
40
41          always @(posedge clk or posedge reset)
42          begin
43                  if (reset) p<=19'b1000000000000000000;
44                  else
45                  begin
46                          p[1]<=p[1]&~t[1]|p[19]&t[15];
47                          p[2]<=p[2]&~t[2]|p[1]&t[1];
48                          p[3]<=p[3]&~t[3]|p[1]&t[1];
49                          p[4]<=p[4]&~t[4]|p[1]&t[1];
50                          p[5]<=p[5]&~t[5]|p[2]&t[2];
51                          p[6]<=p[6]&~t[6]|p[3]&t[3];
52                          p[7]<=p[7]&~t[7]|p[4]&t[4];
53                          p[8]<=p[8]&~t[8]|p[5]&t[5];
54                          p[9]<=p[9]&~t[8]|p[6]&t[6];
55                          p[10]<=p[10]&~t[9]|p[8]&p[9]&t[8];
56                          p[11]<=p[11]&~t[10]|p[10]&t[9];
57                          p[12]<=p[12]&~t[11]|p[10]&t[9];
58                          p[13]<=p[13]&~t[10]|p[7]&t[7];
59                          p[14]<=p[14]&~t[11]|p[7]&t[7];
60                          p[15]<=p[15]&~t[12]|p[11]&p[13]&t[10];
61                          p[16]<=p[16]&~t[13]|p[12]&p[14]&t[11];
62                          p[17]<=p[17]&~t[14]|p[15]&t[12];
63                          p[18]<=p[18]&~t[14]|p[16]&t[13];
64                          p[19]<=p[19]&~t[15]|p[17]&p[18]&t[14];
65                  end
66          end
67  endmodule
```

Lst. C.2. Description of *Mixer*, parallel encoding (after decomposition)

```
1  //Method 2: parallel encoding of the places after
2  //          decomposition of the Petri net
3  //
4  module PN2_ParallelEncoding(y,x,clk,reset);
5          output [1:12] y;
6          input [1:13] x;
7          input reset,clk;
8          reg [1:10] q;
9          wire [1:15] t;
10
11 `define p1  ~q[1]&~q[2]&~q[3]&~q[4]&~q[5]&~q[6]&~q[7]&~q[8]&~q[9]
12 `define p2  q[1]&~q[2]&~q[3]&~q[4]
13 `define p3  q[5]&~q[6]&~q[7]
14 `define p4  ~q[8]&q[9]
15 `define p5  q[1]&q[2]&~q[3]&~q[4]
16 `define p6  q[5]&q[6]&~q[7]
17 `define p7  q[8]&q[9]
18 `define p8  ~q[1]&q[2]&~q[3]&~q[4]
19 `define p9  ~q[5]&q[6]&~q[7]
20 `define p10 ~q[1]&q[2]&q[3]&~q[4]&~q[5]&q[6]&q[7]
21 `define p11 ~q[1]&~q[2]&q[3]&~q[4]
22 `define p12  q[5]&q[6]&q[7]
23 `define p13  q[10]
24 `define p14  q[8]&~q[9]
25 `define p15 ~q[1]&~q[2]&q[3]&q[4]
26 `define p16  q[5]&~q[6]&q[7]
27 `define p17 ~q[1]&~q[2]&~q[3]&q[4]
28 `define p18 ~q[5]&~q[6]&q[7]
29 `define p19 ~q[1]&q[2]&~q[3]&q[4]
30
31         assign t[1]=x[1]&~x[4]&`p1;
32         assign t[2]=x[5]&`p2;
33         assign t[3]=x[7]&`p3;
34         assign t[4]=x[4]&`p4;
35         assign t[5]=x[2]&`p5;
36         assign t[6]=x[3]&`p6;
37         assign t[7]=x[13]&`p7;
38         assign t[8]=`p8&`p9;
39         assign t[9]=~x[6]&~x[8]&~x[9]&`p10;
40         assign t[10]=`p11&`p13;
41         assign t[11]=`p12&`p14;
42         assign t[12]=x[10]&`p15;
43         assign t[13]=x[11]&`p16;
44         assign t[14]=`p17&`p18;
45         assign t[15]=x[12]&`p19;
46         assign y[1]=`p5;
47         assign y[2]=`p6;
48         assign y[3]=`p4;
```

```
49          assign y[4]=`p10;
50          assign y[5]=`p10;
51          assign y[6]=`p10;
52          assign y[7]=`p15;
53          assign y[8]=`p16;
54          assign y[9]=`p19;
55          assign y[10]=`p2;
56          assign y[11]=`p3;
57          assign y[12]=`p7;
58
59      always @(posedge clk or posedge reset) begin
60      if (reset) q<=0;
61      else
62        begin
63          q[1]<=`p1&t[1]|`p2|`p5&~t[5];
64          q[2]<=`p2&t[2]|`p5|`p8|`p10&~t[9]|`p17&t[14]|`p19&~t[15];
65          q[3]<=`p8&t[8]|`p10|`p11|`p15&~t[12];
66          q[4]<=`p11&t[10]|`p15|`p17|`p19&~t[15];
67          q[5]<=`p1&t[1]|`p3|`p6&~t[6]|`p10&t[9]|`p12|`p16&~t[13];
68          q[6]<=`p3&t[3]|`p6|`p9|`p10|`p12&~t[11];
69          q[7]<=`p9&t[8]|`p10|`p12|`p16|`p18&~t[14];
70          q[8]<=`p4&t[4]|`p7|`p14&~t[11];
71          q[9]<=`p1&t[1]|`p4|`p7&~t[7];
72          q[10]<=`p7&t[7]|`p13&~t[10];
73        end
74      end
75  endmodule
```

Lst. C.3. Description of *Mixer*, after decomposition into 4 subnets

```
1  //Method 3: decomposition of the Petri net into 4 subnets (SMCs)
2  //
3  module PN2_Decomposition(y,x,clk,reset);
4           output [1:12] y;
5           input [1:13] x;
6           input reset,clk;
7           wire p1,p7,p8,p9,p10,p11,p12,p13,p14,p17,p18,p20,p22,p21;
8           wire [1:9] q;
9
10 SMC1 a1 (p1,p8,p17,p10,p11,q[1:4],x[1],x[2],x[4],x[5],x[6],x[8],
11          x[9],x[10],x[12],clk,reset,p9,p13,p18,p20,p22,p21);
12 Memory1 m1 (y[1],y[4],y[5],y[6],y[7],y[9],y[10],~clk,reset,
13              q[1:4]);
14 SMC2 a2 (p9,p12,p18,p20,p21,q[5:7],x[1],x[3],x[4],x[6],x[7],x[8],
15          x[9],x[11],clk,reset,p1,p8,p10,p14,p17,p22);
16 Memory2 m2 (y[2],y[8],y[11],~clk,reset,q[5:7]);
17 SMC3 a3 (p7,p14,p22,q[8:9],x[1],x[4],x[13],clk,reset,p1,p12,p20);
18 Memory3 m3 (y[3],y[12],~clk,reset,q[8:9]);
19 SMC4 a4 (p13,x[13],clk,reset,p7,p11);
20 endmodule
```

```
21 module SMC1(p1,p8,p17,p10,p11,q,x1,x2,x4,x5,x6,x8,
22                   x9,x10,x12,clk,reset,p9,p13,p18,p20,p22,p21);
23
24         output p1,p8,p10,p11,p17;
25         output reg [1:4] q;
26         input x1,x2,x4,x5,x6,x8,x9,x10,x12;
27         input reset,clk;
28         input p9,p13,p18,p20,p22,p21;
29
30         wire t1,t2,t5,t8,t9,t10,t12,t14,t15;
31         wire p2,p5,p15,p19;
32
33         assign p1=~q[1]&~q[2]&~q[3]&~q[4];
34         assign p2=q[1]&~q[2]&~q[3]&~q[4];
35         assign p5=q[1]&q[2]&~q[3]&~q[4];
36         assign p8=~q[1]&q[2]&~q[3]&~q[4];
37         assign p10=~q[1]&q[2]&q[3]&~q[4];
38         assign p11=~q[1]&~q[2]&q[3]&~q[4];
39         assign p15=~q[1]&~q[2]&q[3]&q[4];
40         assign p17=~q[1]&~q[2]&~q[3]&q[4];
41         assign p19=~q[1]&q[2]&~q[3]&q[4];
42         assign t1=x1&~x4&p1&p20&p22;
43         assign t2=x5&p2;
44         assign t5=x2&p5;
45         assign t8=p8&p9;
46         assign t9=~x6&~x8&~x9&p10&p21;
47         assign t10=p11&p13;
48         assign t12=x10&p15;
49         assign t14=p17&p18;
50         assign t15=x12&p19;
51
52         always @(posedge clk or posedge reset)
53         begin
54                  if (reset) q<=0;
55                  else
56                  begin
57                    q[1]<=p1&t1|p2|p5&~t5;
58                    q[2]<=p2&t2|p5|p8|p10&~t9|p17&t14|p19&~t15;
59                    q[3]<=p8&t8|p10|p11|p15&~t12;
60                    q[4]<=p11&t10|p15|p17|p19&~t15;
61                  end
62         end
63 endmodule
64
65 module Memory1(y1,y4,y5,y6,y7,y9,y10,clk,reset,q);
66         input clk,reset;
67         input [1:4] q;
68         output y1,y4,y5,y6,y7,y9,y10;
69         reg [1:7] y;
```

```
70 // synthesis attribute bram_map of Memory1 is yes
71         always @(posedge clk)
72         begin
73                 if (reset) y=0;
74                 case (q)
75                         4'b1100:y=7'b1000000;
76                         4'b0110:y=7'b0111000;
77                         4'b0011:y=7'b0000100;
78                         4'b0101:y=7'b0000010;
79                         4'b1000:y=7'b0000001;
80                         default:y=0;
81                 endcase
82         end
83         assign {y1,y4,y5,y6,y7,y9,y10}=y;
84 endmodule
85
86 module SMC2(p9,p12,p18,p20,p21,q,x1,x3,x4,x6,x7,x8,
87                 x9,x11,clk,reset,p1,p8,p10,p14,p17,p22);
88         output p9,p12,p18,p20,p21;
89         input x1,x3,x4,x6,x7,x8,x9,x11;
90         input reset,clk;
91         input p1,p8,p10,p14,p17,p22;
92
93         output reg [5:7] q;
94         wire t1,t3,t6,t8,t9,t11,t13,t14;
95         wire p3,p6,p16,p18;
96
97         assign p3=q[5]&~q[6]&~q[7];
98         assign p6=q[5]&q[6]&~q[7];
99         assign p9=~q[5]&q[6]&~q[7];
100        assign p12=q[5]&q[6]&q[7];
101        assign p16=q[5]&~q[6]&q[7];
102        assign p18=~q[5]&~q[6]&q[7];
103        assign p20=~q[5]&~q[6]&~q[7];
104        assign p21=~q[5]&q[6]&q[7];
105        assign t1=x1&~x4&p1&p20&p22;
106        assign t3=x7&p3;
107        assign t6=x3&p6;
108        assign t8=p8&p9;
109        assign t9=~x6&~x8&~x9&p10&p21;
110        assign t11=p12&p14;
111        assign t13=x11&p16;
112        assign t14=p17&p18;
113
114 always @(posedge clk or posedge reset)
115     begin
116           if (reset) q<=0;
117           else
118           begin
```

```
119                     q[5]<=p1&t1|p3|p6&~t6|p10&t9|p12|p16&~t13;
120                     q[6]<=p3&t3|p6|p9|p10|p12&~t11;
121                     q[7]<=p9&t8|p10|p12|p16|p18&~t14;
122           end
123         end
124 endmodule
125
126 module Memory2(y2,y8,y11,clk,reset,q);
127         input clk,reset;
128         input [5:7] q;
129         output y2,y8,y11;
130
131         reg [1:3] y;
132 // synthesis attribute bram_map of Memory2 is yes
133         always @(posedge clk)
134         begin
135                 if (reset) y=0;
136                 case (q)
137                         3'b110:y=7'b100;
138                         3'b101:y=7'b010;
139                         3'b100:y=7'b001;
140                         default:y=0;
141                 endcase
142         end
143         assign {y2,y8,y11}=y;
144 endmodule
145
146 module SMC3(p7,p14,p22,q,x1,x4,x13,clk,reset,p1,p12,p20);
147         output p7,p14,p22;
148         input x1,x4,x13;
149         input reset,clk;
150         input p1,p12,p20;
151
152         output reg [8:9] q;
153         wire t1,t4,t7,t11;
154         wire p4;
155
156         assign p4=~q[8]&q[9];
157         assign p7=q[8]&q[9];
158         assign p14=q[8]&~q[9];
159         assign p22=~q[8]&~q[9];
160         assign t1=x1&~x4&p1&p20&p22;
161         assign t4=x4&p4;
162         assign t7=x13&p7;
163         assign t11=p12&p14;
164
165         always @(posedge clk or posedge reset)
166         begin
167                 if (reset) q<=0;
```

```
168                     else
169                     begin
170                             q[8]<=p4&t4|p7|p14&~t11;
171                             q[9]<=p1&t1|p4|p7&~t7;
172                     end
173             end
174 endmodule
175
176 module Memory3(y3,y12,clk,reset,q);
177             input clk,reset;
178             input [8:9] q;
179             output y3,y12;
180
181             reg [1:2] y;
182 // synthesis attribute bram_map of Memory3 is yes
183             always @(posedge clk)
184             begin
185                     if (reset) y=0;
186                     case (q)
187                             2'b01:y=7'b10;
188                             2'b11:y=7'b01;
189                             default:y=0;
190                     endcase
191             end
192             assign {y3,y12}=y;
193 endmodule
194
195 //4th SMC does not have any output thus there is no 4th memory
196 module SMC4(p13,x13,clk,reset,p7,p11);
197             output p13;
198             input x13;
199             input reset,clk;
200             input p7,p11;
201
202             reg q;
203             wire t7,t10;
204             wire p23;
205
206             assign p13=q;
207             assign p23=~q;
208             assign t7=x13&p7&p23;
209             assign t10=p11&p13;
210
211             always @(posedge clk or posedge reset)
212             begin
213                     if (reset) q<=0;
214                     else q<=p7&t7|p13&~t10;
215             end
216 endmodule
```

C.2. Structural Decomposition of a Discrete System

Lst. C.4. The description of the initial memory

```
1  //Description of the initial memory:
2  module Initial_memory (y,clk,reset,address);
3          output reg [1:6] y;
4          input clk,reset;
5          input [1:2] address;
6
7  // synthesis attribute bram_map of Initial_memory is yes
8          always @(posedge clk)
9          begin
10                 if (reset) y=6'b 010001;
11                 case (address)
12                         2'b00:y=6'b      010001;
13                         2'b01:y=6'b      010100;
14                         2'b10:y=6'b      100010;
15                         2'b11:y=6'b      001010;
16                 endcase
17         end
18 endmodule
```

Lst. C.5. The description of the memory after the structural decomposition

```
1  // The description of the decomposed memory:
2  module Decomposed_memory (y,clk,reset,address);
3          output [1:6] y;
4          input clk,reset;
5          input [1:2] address;
6          wire [1:4] q;
7
8          Memory_block mem (q,clk,reset,address);
9
10         //decoding of the microoperation:
11         assign y[1]=~q[1]&q[2];
12         assign y[2]=~q[1]&~q[2];
13         assign y[3]=q[1]&~q[2];
14         assign y[4]=~q[3]&q[4];
15         assign y[5]=q[3]&~q[4];
16         assign y[6]=~q[3]&~q[4];
17 endmodule
18
19 module Memory_block (y,clk,reset,address); // memory block
20         output reg [1:4] y;
21         input clk,reset;
```

```
22          input [1:2] address;
23
24  // synthesis attribute bram_map of Memory_block is yes
25          always @(posedge clk)
26          begin
27                  if (reset) y=4'b 0000;
28                  case (address)
29                          2'b00:y=4'b     0000;
30                          2'b01:y=4'b     0001;
31                          2'b10:y=4'b     0110;
32                          2'b11:y=4'b     1010;
33                  endcase
34          end
35  endmodule
```

Bibliography

Acharya, B. D. and Acharya, M. (2003). *A new characterisation of connected hypergraphs*, Electronic Notes in Discrete Mathematics, Vol. 15, pp. 6–9.

Adamski, M. (1990). *Application of a Systematic Structural Method to Design of the Digital Circuits*, Higher School of Engineering Publishing House, Zielona Góra (in Polish).

Adamski, M. (1991). *Parallel controller implementation using standard PLD software*, FPGAs: International Workshop on Field Programmable Logic and Applications, Abingdon EE&CS Books, Abingdon, pp. 296–304.

Adamski, M. and Barkalov, A. (2006). *Architectural and Sequential Synthesis of Digital Devices*, University of Zielona Góra Press, Zielona Góra.

Adamski, M. and Chodań, M. (2000). *Modeling of Discrete Control Systems with the Use of SFC Nets*, Technical University of Zielona Góra Publishing House, Zielona Góra (in Polish).

Adamski M., Karatkevich A. and Węgrzyn M. (eds) (2005). *Design of Embedded Control Systems*, Springer, New York, NY, USA.

Adamski, M. and Węgrzyn, M. (2009). *Petri nets mapping into reconfigurable logic controllers*, Electronics and Telecommunications Quarterly, Vol. 55, No. 2, pp. 157–182.

Adamski, M. and Wiśniewska, M. (2006). *Usage of hypergraphs in decomposition of concurrent automata*, Measurement Automation and Monitoring, Vol. 52, No. 6, pp. 8–10 (in Polish).

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, USA.

Banaszak, Z., Kuś, J. and Adamski, M. (1993). *Petri Nets: Modeling, Control and Synthesis of Discrete Systems*, Higher School of Engineering Publishing House, Zielona Góra.

Banaszak, Z., Majdzik, P. and Wójcik, R. (2008). *Concurrent Processes. The Effectiveness of the Functional Models*, Technical University of Koszalin Publishing House, Koszalin.

Baranov, S. I. (1994). *Logic Synthesis for Control Automata*, Kluwer Academic Publishers, Boston, MA, USA.

Barkalov, A. and Titarenko, L. (2009). *Logic Synthesis for FSM-based Control Units*, Lecture Notes in Electrical Engineering, Vol. 53, Springer-Verlag, Berlin.

Berge, C. (1973). *Graphs and Hypergraphs*, American Elsevier Pub. Co., New York, NY, USA.

Berge, C. (1989). *Hypergraphs: Combinatorics of Finite Sets*, North-Holland, Amsterdam.

Bilinski, K., Adamski, M., Saul, J. and Dagless, E. (1994). *Petri-net-based algorithms for parallel-controller synthesis*, IEE Proceedings - Computers and Digital Techniques, Vol. 141, No. 6, pp. 405–412.

Blanchard, M. (1979). *Comprendre, Maîtriser Et Appliquer Le Grafcet*, Cepadues, Toulouse (in French).

Brayton, R. K., Sangiovanni-Vincentelli, A. L., McMullen, C. T. and Hachtel, G. D. (1984). *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Norwell, MA, USA.

Brown, S. and Vernesic, Z. (2000). *Fundamentals of Digital Logic with VHDL Design*, McGraw Hill, New York, NY, USA.

Bukowiec, A. (2009). *Synthesis of Finite State Machines for FPGA Devices Based on Architectural Decomposition*, Lecture Notes in Control and Computer Science, Vol. 13, University of Zielona Góra Press, Zielona Góra.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms*, The MIT Press, Cambridge, MA, USA.

Corno, F., Prinetto, P. and Sonza Reorda, M. (1995). *Using symbolic techniques to find the maximum clique in very large sparse graphs*, EDTC '95: Proceedings of the 1995 European conference on Design and Test, IEEE Computer Society, Washington, DC, USA, p. 320.

Czumaj, A. and Sohler, C. (2005). *Testing hypergraph colorability*, Theor. Comput. Sci., Vol. 331, No. 1, pp. 37–52.

Dasgupta, S. (1979). *The organization of microprogram stores*, ACM Comput. Surv., Vol. 11, No. 1, pp. 39–65.

David, R. and Alla, H. (1992). *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

DeMicheli, G. (1994). *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Higher Education.

Doligalski, M. and Adamski, M. (2010). *The specification of digital controllers oriented towards reliability*, Measurement Automation and Monitoring, Vol. 56, No. 7, pp. 671–674 (in Polish).

Eiter, T. (1994). *Exact transversal hypergraphs and application to boolean μ-functions*, Journal of Symbolic Computation, Vol. 17, No. 3, pp. 215–225.

Eiter, T. and Gottlob, G. (1995). *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput., Vol. 24, No. 6, pp. 1278–1304.

Eiter, T. and Gottlob, G. (2002). *Hypergraph transversal computation and related problems in logic and AI*, JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence, Springer-Verlag, London, UK, pp. 549–564.

Elbassioni, K. and Rauf, I. (2010). *Polynomial-time dualization of r-exact hypergraphs with applications in geometry*, Discrete Mathematics, Vol. 310, No. 17-18, pp. 2356 – 2363.

Fernandes, J., Adamski, M. and Proença, A. (1997). *VHDL generation from hierarchical Petri net specifications of parallel controllers*, IEE Proceedings - Computers and Digital Techniques, Vol. 144, No. 2, pp. 127–137.

Franklin, M. and Saluja, K. K. (1994). *Hypergraph coloring and reconfigured RAM testing*, IEEE Trans. Comput., Vol. 43, No. 6, pp. 725–736.

Girault, C. and Valk, R. (2003). *Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications*, Springer.

Harary, F. (1994). *Graph Theory*, Addison-Wesley, Reading, MA, USA.

Hong, S.-K., Park, I.-C. and Kyung, C.-M. (1990). *An $o(^3log)$-heuristic for microcode bit optimization*, ICCAD, Santa Clara, CA, USA, pp. 180–183.

Hoos, H. H. and Stützle, T. (2000). *Local search algorithms for SAT: an empirical evaluation*, J. Autom. Reasoning, Vol. 24, No. 4, pp. 421–481.

Kania, D. (2004). *The Logic Synthesis for the PAL-based Complex Programmable Logic Devices*, Lecture Notes of the Silesian University of Technology, Gliwice.

Kania, D. and Kulisz, J. (2007). *Logic synthesis for PAL-based CPLD-s based on two-stage decomposition*, J. Syst. Softw., Vol. 80, No. 7, pp. 1129–1141.

Karatkevich, A. (2007). *Dynamic Analysis of Petri Net-based Discrete Systems*, Lecture Notes in Control and Information Sciences, 356, Springer-Verlag, Berlin.

Karatkevich, A. (2008). *On reduction of SM-subnets in Petri nets*, Telecommunication Review and Telecommunication News, Vol. 6, pp. 806–808 (in Polish).

Kavvadias, D. and Stavropoulos, E. C. (1999). *Evaluation of an algorithm for the transversal hypergraph problem*, Proceedings of the 3rd International Workshop on Algorithm Engineering, Springer-Verlag, London, UK, pp. 72–84.

Kernighan, B. W. and Ritchie, D. M. (1977). *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, USA.

Knuth, D. (1997). *The Art of Computer Programming*, 3^{rd} edn, Addison-Wesley, Reading, MA, USA.

Knuth, D. (2000). *Dancing links*, Millennial Perspectives in Computer Science, Palgrave, pp. 187–214.

Konstantinova, E. V. and Skorobogatov, V. A. (2001). *Application of hypergraph theory in chemistry*, Discrete Mathematics, Vol. 235, No. 1-3, pp. 365 – 383.

Korzan, B. (1978). *Elements of the Theory of Graphs and Networks*, WNT, Warszawa (in Polish).

Kovalyov, A. (1992). *Concurrency relations and the safety problem for Petri nets*, Proc. 13th Int. Conf. on Application and Theory of Petri nets, Sheffield, UK, pp. 299–309.

Kozłowski, T., Dagless, E., Saul, J., Adamski, M. and Szajna, J. (1995). *Parallel controller synthesis using Petri nets*, IEE Proceedings - Computers and Digital Techniques, Vol. 142, No. 4, pp. 263–271.

Krivelevich, M. and Sudakov, B. (1998). *Approximate Coloring of Uniform Hypergraphs*, Technical report, ESA '98 Proceedings of the 6th Annual European Symposium on Algorithms.

Kubale, M. (2002). *Discrete Optimization. Models and Methods for Graphs Coloring*, WNT, Warszawa (in Polish).

Kubale, M., Obszarski, P. and Piwakowski, K. (2006). *Hypergraphs coloring*, Technical Reports of the Silesian Technical University, Vol. 143, pp. 83–90 (in Polish).

Łabiak, G. (2005). *Application of Hierarchical Model of Concurrent Automaton in Digital Controller Design*, Lecture Notes in Control and Computer Science, Vol. 6, University of Zielona Góra Press, Zielona Góra (in Polish).

Lee-Kwang, H. and Cho, C. H. (1996). *Hierarchical reduction and partition of hypergraph*, IEEE Transitions on systems, Vol. 26, No. 2, pp. 340–344.

Leinweber, L. and Bhunia, S. (2008). *Fine-grained supply gating through hypergraph partitioning and Shannon decomposition for active power reduction*, DATE '08: Proceedings of the conference on Design, Automation and Test in Europe, ACM, New York, NY, USA, pp. 373–378.

Lovász, L. (1972). *Normal hypergraphs and the weak perfect graph conjecture*, discmath, Vol. 2, pp. 253–267.

Lovász, L. (1973). *Coverings and colorings of hypergraphs*, Proceedings of the 4th Southeastern Conf. on Combinatorics, Graph Theory and Computing, Utilitas Math., Winnipeg, Man., pp. 3–12.

Łuba, T. (2005). *Synthesis of Logic Devices*, Warsaw University of Technology Press, Warszawa (in Polish).

Malykh, O. N. and Shakhnovskii, Y. S. (2004). *Optimization of a data dependence graph for the local microcode compaction problem. Part 2: algorithms and experimental verification*, Programming and Computer Software, Vol. 30, pp. 134–141.

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Academic Press, Inc., Orlando, FL, USA.

McDiarmid, C. (1997). *Hypergraph colouring and the Lovász local lemma*, Discrete Math., Vol. 167-168, pp. 481–486.

Mealy, G. (1955). *A method for synthesizing sequential circuits*, BSTJ, Vol. 34, pp. 1045–1079.

Mielcarek, K. (2009). *Application of Perfect Graphs in Designing of Digital Circuits*, PhD thesis, University of Zielona Góra, Zielona Góra (in Polish).

Moore, E. (1956). *Gedanken experiments on sequential machines*, Automata Studies, Princeton University Press, pp. 129–153.

Murata, T. (1989). *Petri nets: properties, analysis and applications*, Proceedings of the IEEE, Vol. 77, No. 4, pp. 541–580.

Papachristou, C. A. (1979). *A scheme for implementing microprogram addressing with programmable logic arrays*, Digital Processes, Vol. 5, No. 3-4, pp. 235–256.

Papadimitriou, H. C. (1994). *Computational Complexity*, Addison-Wesley, Reading, MA, USA.

Pardey, J., Amroun, A., Bolton, M. and Adamski, M. (1994). *Parallel controller synthesis for programmable logic devices*, Microprocessors and Microsystems, Vol. 18, No. 8, pp. 451–457.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR, Upper Saddle River, NJ, USA.

Petri, C. A. (1962). *Kommunikation Mit Automaten*, PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany (in German).

Puri, R., Member, S. and Member, S. (1993). *Microword length minimization in microprogrammed controller synthesis*, IEEE Transactions on Computer-Aided Design, Vol. 12, pp. 1449–1457.

Rawski, M., Selvaraj, H. and Łuba, T. (2003). *An application of functional decomposition in ROM-based FSM implementation in FPGA devices*, DSD '03: Proceedings of the Euromicro Symposium on Digital Systems Design, IEEE Computer Society, Washington, DC, USA, p. 104.

Robertson, E. L. (1979). *Microcode bit optimization is NP-complete*, IEEE Trans. Comput., Vol. 28, No. 4, pp. 316–319.

Roguska, A. (2001). *Evaluation of the Practical Interpreted Petri Nets, Grafcet Networks and Networks Based On the SEC Binary Control System Design*, Bachelor's thesis, Technical University of Zielona Góra, Zielona Góra (in Polish).

Rudell, R. L. (1989). *Logic Synthesis for VLSI Design*, PhD thesis, EECS Department, University of California, Berkeley, CA, USA.

Sapiecha, P. (1999). *Synthesis Algorithms for Boolean Functions and Relations in Terms of Methods of Representation and Data Compression*, PhD thesis, Warsaw University of Technology, Warszawa (in Polish).

Sasao, T. (1999). *Totally undecomposable functions: applications to efficient multiple-valued decompositions*, ISMVL '99: Proceedings of the 29^{th} IEEE International Symposium on Multiple-Valued Logic, IEEE Computer Society, Washington, DC, USA, p. 59.

Sentovich, E., Singh, K. J., Moon, C. W., Savoj, H., Brayton, R. K. and Sangiovanni-Vincentelli, A. L. (1992). *Sequential circuit design using synthesis and optimization*, ICCD '92: Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors, IEEE Computer Society, Washington, DC, USA, pp. 328–333.

Shi, C.-J. and Brzozowski, J. A. (1992). *Efficient constrained encoding for VLSI sequential logic synthesis*, EURO-DAC '92: Proceedings of the conference on European design automation, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 266–271.

Steiglitz, K. (1974). *Introduction to Discrete Systems*, John Wiley, New York, NY, USA.

Stroustrup, B. (1986). *The C++ Programming Language*, Addison-Wesley, Reading, MA, USA.

Sysło, M. M., Deo, N. and Kowalik, J. S. (1983). *Discrete Optimization Algorithms: with Pascal Programs*, Prentice-Hall.

Szpyrka, M. (2008). *Petri Nets in Modeling and Analysis of Concurrent Systems*, WNT, Warszawa (in Polish).

Thomas, D. and Moorby, P. (2002). *The Verilog Hardware Description Language*, 5^{th} edn, Kluwer Academic Publishers, Norwell, MA, USA.

Tkacz, J. (2006). *Gentzen system calculus implementation for symbolic minimalization of complicated logical expressions*, Discrete-Event System Design, DESDes '06: Proceedings volume from the 3^{rd} IFAC Workshop, University of Zielona Góra Press, Zielona Góra, pp. 53–56.

Tuza, Z. and Voloshin, V. (2000). *Uncolorable mixed hypergraphs*, Discrete Appl. Math., Vol. 99, No. 1-3, pp. 209–227.

Tuza, Z., Voloshin, V. and Zhou, H. (2002). *Uniquely colorable mixed hypergraphs*, Discrete Math., Vol. 248, No. 1-3, pp. 221–236.

Valette, R. (1978). *Comparative study of switching representation tool with GRAFCET and Petri nets*, Nouv. Autom., Vol. 23, No. 12, pp. 377–382.

Wahlström, M. (2004). *Exact algorithms for finding minimum transversals in rank-3 hypergraphs*, J. Algorithms, Vol. 51, No. 2, pp. 107–121.

Węgrzyn, M., Adamski, M. and Monteiro, J. (1996). *VHDL simulation of Xilinx-FPGA-based concurrent controller*, Proc. of a workshop on application of programmable logic, Lisbon, Portugal, pp. 12–15.

Wilkes, M. V. (1951). *The best way to design an automatic calculating machine*, Manchester University Inaugural Conference, Manchester, UK.

Wilson, R. J. (1979). *Introduction to Graph Theory*, Academic Press, New York, NY, USA.

Wiśniewska, M. (2005). *Decomposition of Petri nests to subnets based on the hypergraphs*, Proc. of the VII International PHD Workshop - OWD 2005, Gliwice, Wisła, pp. 81–84.

Wiśniewska, M. (2009). *Microinstruction length reduction in design of microprogrammed controllers*, Measurement Automation and Monitoring, Vol. 55, No. 8, pp. 575–577 (in Polish).

Wiśniewska, M. (2011). *CAD system for automatic decomposition of discrete systems based on hypergraphs*, Proc. of the Computer Science - KNWS' 11: Proceedings of the 8^{th} conference, Karpacz, pp. 17–20.

Wiśniewska, M. and Adamski, M. (2006). *Hypergraphs in the state space analysis of concurrent automata described by Petri nets*, Measurement Automation and Monitoring, Vol. 52, No. 7, spec. ed., pp. 62–64 (in Polish).

Wiśniewska, M. and Adamski, M. (2008a). *Usage of hypergraph duality in decomposition of concurrent automata*, Telecommunication Review and Telecommunication News, Vol. 6, pp. 731–733 (in Polish).

Wiśniewska, M. and Adamski, M. (2008b). *Usage of hypergraphs in hierarchical decomposition of discrete system*, Measurement Automation and Monitoring, Vol. 54, No. 8, pp. 517–519 (in Polish).

Wiśniewska, M., Adamski, M. and Wiśniewski, R. (2011). *Application of the hypergraph theory in the selection of compatibility classes*, Proc. of the Computer Science - KNWS' 11: Proceedings of the 8^{th} conference, Karpacz, pp. 11–15.

Wiśniewska, M. and Wiśniewski, R. (2005). *Reduction of the memory volume of microprogrammed controllers with usage of hypergraphs*, Proc. of the Computer Science - KNWS' 05: Proceedings of the 2^{nd} conference, University of Zielona Góra Press, Złotniki Lubańskie, pp. 23–32.

Wiśniewska, M. and Wiśniewski, R. (2010). *Application of hypergraph coloring in decomposition of concurrent automata*, Methods of Applied Computer Science, Vol. 23, No. 2, pp. 151–157 (in Polish).

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2005). *Application of hypergraphs to optimization of the microinstruction length in microprogrammed controllers*, Proc. of the Computer Science - RUC 2005: 8^{th} conference on Reprogrammable Digital Circuits, Technical University of Szczecin, Szczecin, pp. 33–40.

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2006). *Usage of hypergraphs in decomposition of concurrent automata*, 1st. International Conference for Young Researchers in Computer Science, Control, Electrical Engineering and Telecommunications - ICYR : abstracts, University of Zielona Góra Press, Zielona Góra, pp. 58–59.

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2007a). *Usage of hypergraphs in decomposition of discrete system*, Measurement Automation and Monitoring, Vol. 53, No. 5, pp. 129–131 (in Polish).

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2007b). *Usage of hypergraph theory in decomposition of concurrent automata*, Measurement Automation and Monitoring, Vol. 53, No. 7, pp. 66–68.

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2009a). *Reduction of the microinstruction lenght in the designing process of microprogrammed controllers*, Electrical Review, Vol. 85, No. 7, pp. 203–206.

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2010). *Application of hypergraph transversals in minimization of the memory size*, Measurement Automation and Monitoring, Vol. 56, No. 7, pp. 777–779 (in Polish).

Wiśniewska, M., Wiśniewski, R., Adamski, M. and Halang, W. (2009b). *Application of hypergraphs in microcode lenght reduction of microprogrammed controllers*, Second International Workshop on Nonlinear Dynamics and Sychronization - INDS '09, Aachen, Shaker Verlag, Klagenfurt, Austria, Vol. Smart System Technologies, Bd. 4, pp. 106–109.

Wiśniewski, R., M.Wiśniewska and M.Adamski (2012). *A polynomial algorithm to compute the concurrency hypergraph in Petri nets*, Measurement Automation and Monitoring, Vol. 58, No. 7, pp. 650–652 (in Polish).

Wiśniewski, R. (2009). *Synthesis of Compositional Microprogram Control Units for Programmable Devices*, Lecture Notes in Control and Computer Science, Vol. 14, University of Zielona Góra Press, Zielona Góra.

Zakrevskii, A. D. (1986). *Petri nets modeling of logical control algorithms*, Automatic Control and Computer Sciences, Vol. 20, No. 6, pp. 38–45.

Zakrevskij, A. D., Karatkevich, A. and Adamski, M. (2002). *A method of analysis of operational Petri nets*, Advanced computer systems : Eight International Conference, ACS 2001, ISBN: 0-7923-7651-X, Kluwer Academic Publishers, Boston, MA, USA, pp. 449–460.

Zwolinski, M. (2000). *Digital System Design with VHDL*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

List of figures

List of tables

List of listings

Dekompozycja systemów dyskretnych z wykorzystaniem hipergrafów

Streszczenie

W ostatnich latach zaobserwować można dynamiczny rozwój technologii elektronicznych oraz informatycznych. Wiąże się to ze wzrostem realizowanych systemów dyskretnych, a co za tym idzie, ciągłym udoskonaleniem metod oraz narzędzi projektowych. Głównym celem opracowywanych algorytmów jest uzyskanie satysfakcjonujących wyników w możliwie jak najkrótszym czasie. Zdecydowana większość zadań jest współcześnie realizowana z wykorzystaniem klasycznej teorii grafów nieskierowanych, które jeszcze kilka lat temu w zupełności spełniały stawiane cele. Jednakże wzrost rozmiaru systemów dyskretnych wymusza ciągłą modyfikację istniejących i poszukiwanie nowych metod dekompozycji, analizy czy też projektowania. Swoistym przełomem może okazać się wprowadzenie teorii hipergrafów. Pomimo tego, że hipergrafy zostały zaproponowane już w latach 70 ubiegłego stulecia, dopiero w ostatnich latach coraz śmielej przebijają się zarówno w zagadnieniach związanych z matematyką, jak i informatyką, czy też elektroniką. Niemniej, bardzo ciekawe własności tych struktur powodują intensywny wzrost zainteresowania badaczy na całym świecie, co z kolei wiąże się z praktycznym zastosowaniem hipergrafów w rozwiązywaniu problemów z różnych dziedzin, takich jak matematyka, chemia, a przede wszystkim informatyka i elektronika.

W niniejszej monografii przedstawiono dwie nowe metody dekompozycji systemów dyskretnych. Oba rozwiązania bazują na zastosowaniu teorii hipergrafów. Pierwsza z metod wiąże się z dekompozycją równoległą automatów współbieżnych, opisanych sieciami Petriego. Sterownik dzielony jest na współpracujące ze sobą automaty sekwencyjne, przy czym każdy z nich może zostać zaprojektowany niezależnie, np. jako układ sterujący z pamięcią. Drugie z proponowanych rozwiązań dotyczy dekompozycji strukturalnej systemu dyskretnego. W procesie dekompozycji zmniejszana jest pojemność bloku pamięci (w szczególności bloków pamięci automatów sekwencyjnych), poprzez zastosowaną metodę redukcji rozmiaru mikroinstrukcji.

Głównym celem opracowanych metod jest usprawnienie procesu dekompozycji, poprzez zredukowanie czasu wykonania algorytmu (zarówno dekompozycja równoległa, jak i strukturalna), a także znalezienie skuteczniejszego rozwiązania (w przypadku dekompozycji strukturalnej, skuteczność rozumiana jest poprzez uzyskanie mniejszego rozmiaru dekomponowanej pamięci), w porównaniu do rozwiązań obecnie stosowanych.

Lecture Notes in Control and Computer Science

Editor-in-Chief: Józef KORBICZ

Vol. 23: Monika Wiśniewska
Application of Hypergraphs in Decomposition of Discrete Systems
143 p. 2012 [978-83-7842-025-5]

Vol. 22: Mariusz Jacyno
Self-organising Agent Communities for Autonomic Computing
202 p. 2012 [978-83-7842-012-5]

Vol. 21: Błażej Cichy
Analysis and Control of Multi-dimensional (nD) Spatio-temporal Systems with Non-causal Spatial Variables
151 p. 2012 [978-83-7842-013-2]

Vol. 20: Michał Doligalski
Behavioral Specification Diversification of Reconfigurable Logic Controllers
108 p. 2012 [978-83-7842-005-7]

Vol. 19: Grzegorz Bazydło
Graphic Specification of Programs for Reconfigurable Logic Controllers Using Unified Modeling Language
121 p. 2012 [978-83-7481-470-6]

Vol. 18: Marek Sawerwain
Selected Topics in Quantum Programming Languages Theory
252 p. 2011 [978-83-7481-457-7]

Vol. 17: Jacek Bieganowski
Synthesis of Microprogram Control Units Oriented Toward Decreasing the Number of Macrocells of Addressing Circuit
105 p. 2011 [978-83-7481-454-6]

Vol. 16: Łukasz Dziekan
Neuro-Fuzzy-Based Takagi-Sugeno Modelling in Fault-Tolerant Control
161 p. 2011 [978-83-7481-427-0]

Vol. 15: Łukasz Hładowski
Efficient Algorithms for Solving Large-scale Computational Control Problems of Repetitive Processes
165 p. 2011 [978-83-7481-415-7]

Vol. 14: Remigiusz Wiśniewski
Synthesis of Compositional Microprogram Control Units for Programmable Devices
153 p. 2009 [978-83-7481-293-1]

Vol. 13: Arkadiusz Bukowiec
Synthesis of Finite State Machines for FPGA Devices Based on Architectural Decomposition
102 p. 2009 [978-83-7481-257-3]

Vol. 12: Małgorzata Kołopieńczyk
Application of Address Converter for Decreasing Memory Size of Compositional Microprogram Control Unit with Code Sharing
96 p. 2008 [83-7481-033-5]

Vol. 11: Bartłomiej Sulikowski
Computational Aspects in Analysis and Synthesis of Repetitive Processes
168 p. 2006 [83-7481-033-5]

Vol. 10: Bartosz Kuczewski
Computational Aspects of Discrimination between Models of Dynamic Systems
158 p. 2006 [83-7481-030-0]

Vol. 9: Marek Kowal
Optimization of Neuro-Fuzzy Structures in Technical Diagnostics Systems
116 p. 2005 [83-89712-88-1]

Vol. 8: Wojciech Paszke
Analysis and Synthesis of Multidimensional System Classes Using Linear Matrix Inequality Methods
188 p. 2005 [83-89712-81-4]

Vol. 7: Piotr Steć
Segmentation of Colour Video Sequences Using the Fast Marching Method
110 p. 2005 [83-89712-47-4]

Vol. 6: Grzegorz Łabiak
The Use of Hierarchical Model of Concurrent Automaton in Digital Controller Design (in Polish)
168 p. 2005 [83-89712-42-3]

Vol. 5: Maciej Patan
Optimal Observation Strategies for Parameter Estimation of Distributed Systems
220 p. 2004 [83-89712-03-2]

Vol. 4: Przemysław Jacewicz
Model Analysis and Synthesis of Complex Physical Systems Using Cellular Automata
134 p. 2003 [83-89321-67-X]

Vol. 3: Agnieszka Węgrzyn
Symbolic Analysis of Binary Control Circuits Using Selected Methods of Petri Net Analysis (in Polish)
125 p. 2003 [83-89321-54-8]

Vol. 2: Grzegorz Andrzejewski
Program Model of Interpreted Petri Net for Digital Microsystem Design (in Polish)
109 p. 2003 [83-89321-53-X]

Vol. 1: Marcin Witczak
Identification and Fault Detection of Non-Linear Dynamic Systems
124 p. 2003 [83-88317-65-2]