Boolean networks for modeling and analysis of gene regulation

Dao Zhou, Christoph Müssel, Ludwig Lausser,
Martin Hopfensitz, Michael Kühl, Hans A. Kestler

Ulmer Informatik-Berichte

Nr. 2009-10
Oktober 2009

Boolean Networks for Modeling and Analysis of Gene Regulation

Dao Zhou[1,2,†], Christoph Müssel[1,†], Ludwig Lausser[1,†], Martin Hopfensitz[3], Michael Kühl[4], Hans A. Kestler[1,3*]

[1] Institute of Neural Information Processing, Research Group of Bioinformatics and Systems Biology, University of Ulm, Germany
[2] Hubei Bioinformatics and Molecular Imaging Key Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China
[3] Internal Medicine I, University Hospital Ulm, Germany
[4] Institute for Biochemistry and Molecular Biology, University of Ulm, Germany
† contributed equally

Gene-regulatory networks control the expression of genes and therefore the phenotype of cells. Modeling and simulation of such networks can provide deep insights into the functioning of cells. Boolean networks are a commonly used technique to model gene-regulatory networks. We introduce methods to construct Boolean networks from literature knowledge and to analyze their dynamics. In particular, methods to identify and analyze attractors are presented. In simulations on three biological networks, we analyze the robustness of attractors. These evaluations confirm the biological relevance of previously identified attractors.

1 Background

1.1 Gene-regulatory networks

Anatomy and physiology of all living systems are mainly determined by their genes. Strands of desoxyribonucleic acids (DNA) are stored in the nucleus of nearly all cells and determine their characteristics through the process of gene expression. A DNA chain is a sequence of four chemical bases, adenine (A), cytosine (C), guanine (G) and thymine (T), a sugar molecule (desoxyribose), and phosphate bridges. The DNA can be subdivided into *genes*, which code for hereditary traits. Genes generally consist of coding regions that determine the proteins which make up the cell structure, and of non-coding regions.

[*]hans.kestler@uni-ulm.de

Gene expression is the process of building proteins from the DNA. It comprises two major steps: First, a copy of a piece of DNA is assembled as messenger RNA (mRNA), which is similar to DNA besides the replacement of thymine by another base called uracil. Furthermore, desoxyribose is replaced by ribose. This step is called *transcription*. In a second step, the *translation*, proteins are synthesized from the mRNA.

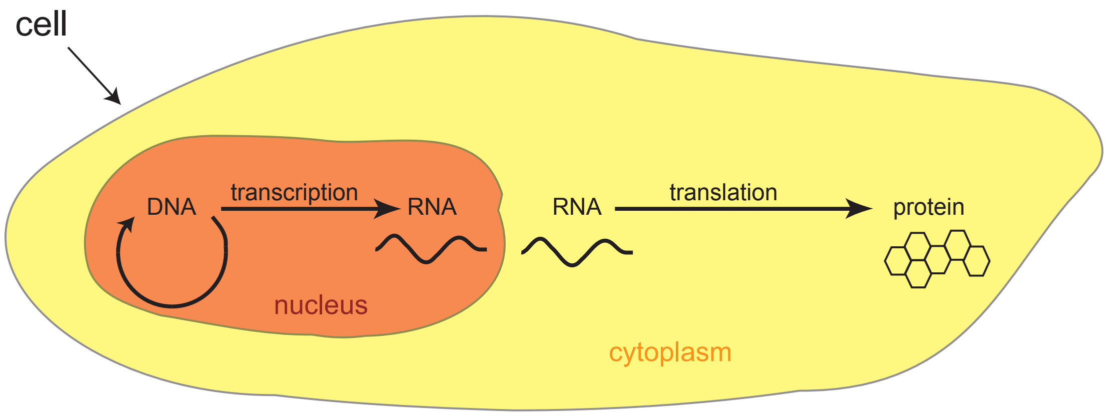

Figure 1: Schematic depiction of gene expression in a cell. Transcription takes place in the nucleus, translation is performed by other parts of the cell.

In each cell, only a small portion of the genes are expressed depending on the cell type, the status, and external conditions. This is controlled by *gene regulation*: The availability of a protein depends on its stability and the number of corresponding mRNA templates. In turn, the amount of mRNA depends on the *regulatory region* dedicated to the corresponding gene, and on the presence of *transcription factors* that activate or suppress this gene. These transcription factors are proteins and thus gene products themselves. Hence, gene expression is a regulatory process: Genes and their expression products influence the expression of other genes and, as a whole, form complex *regulatory networks*. In addition, gene expression in a cell can be influenced by external signals from other cells. In this case, external signal molecules are received and translated to cellular responses, which is known as *signal transduction*. Interactions between multiple cells can make up complex *signalling networks*. More details on the biological background can be found in [2].

Reconstructing such regulatory networks provides a deep insight into the processes inside cells and enables the development of new treatments for diseases that are caused by misregulated gene expression [10]. In addition, simulations with computational models can replace costly biological experiments [12]. To represent the dynamics of such a network, a set of measurements of the corresponding genes over time is required. To be able to simulate as many different network configurations as possible, the measurements should also contain as many activation states of each of the genes as possible. That means that additional experiments with changed environmental conditions or over-expressed and knocked-out genes have to be conducted for each of the time steps [11, 17].

The measurement of the genes poses some major problems: For some network configurations that might occur in theory, it may not be possible to retrieve meaningful biological results, as they never occur in a living cell in practice. In addition, it can be hard to measure genes at sufficiently small time steps, and thus important changes in activation levels may never be visible in the reconstructed networks.

After a set of interactions between gene products has been identified by experiments, researchers can construct mathematical models of the network. Common types of models are:

- Sets of *ordinary differential equations (ODEs)* are widely used to describe interactions between gene products quantitatively [4]. ODEs provide a continuous representation of the network. However, they require time-consuming calculations, as it is usually not possible to solve them analytically. In addition, precise knowledge of a large set of parameters is required [5].

- *Boolean networks* were first proposed for gene-regulatory networks by Kauffman in 1969 [13]. They only allow the states *on* and *off* for a gene. Each gene is represented by a Boolean function over all other involved genes in the previous time step. This is an abstract, but intuitive representation of interactions. In addition, Boolean networks approximate the real nature of gene-regulatory networks well, while being of simple structure: It is assumed that concentration levels in gene-regulatory networks behave like the Hill function [4]. Boolean functions approximate the sigmoidal behaviour of this function by the dichotomous step function (see Fig. 2). Boolean networks allow for the identification of steady states and cycles [5, 12]. Another major advantage of Boolean networks is the fact that natural-language statements from literature can easily be transferred into this representation. For example, the statement "Gene 1 inhibits Gene 2“ can be interpreted as $Gene_2(t+1) = \text{NOT } Gene_1(t)$.
  In addition, the construction of Boolean networks is computationally less costly than the calculations involved in the ODE models [3].

- *Probabilistic Boolean networks* constitute a generalization of Boolean networks to overcome the deterministic rigidity of Boolean networks, and were proposed by Shmulevich et al. [19, 20]. Due to biological uncertainty, experimental noise, or incomplete understanding of a system, the need for uncertainty in the regulatory logic is raised. Probabilistic Boolean networks address this issue by extending the Boolean network model in a way that each target gene can have several candidate functions. Each function is assigned a probability, based on its compatibility with experimental data. The probability is determined by employing the *Coefficient Of Determination* (COD) [6]. In practice, the COD must be estimated from training data using approximations of the candidate functions. Thus, problems arise from the required amount of training data and the complexity of the candidate functions.

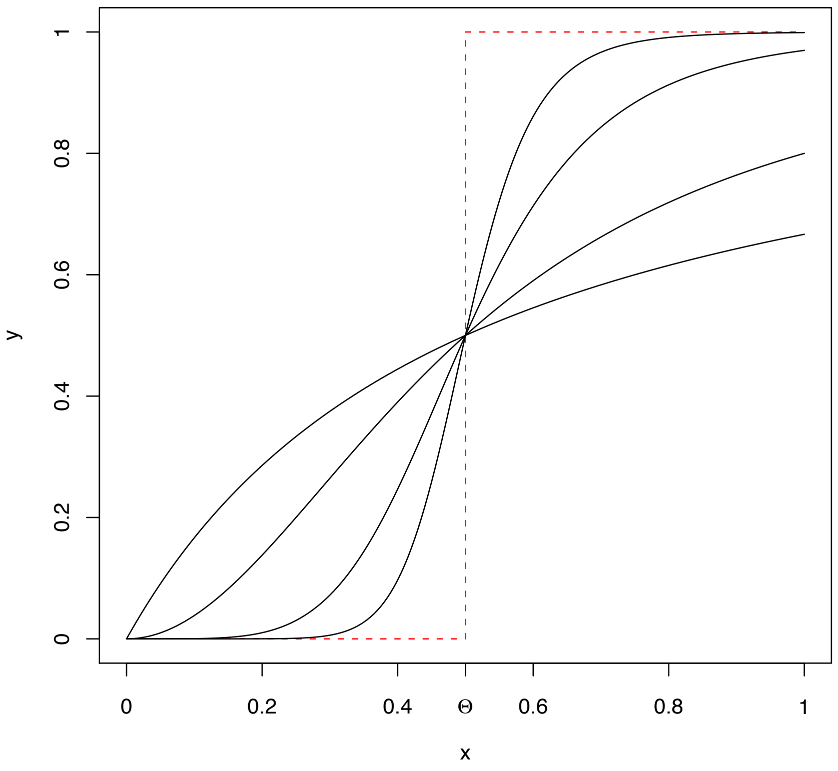

Figure 2: The Hill function $h(x, \Theta, m) = \frac{x^m}{x^m + \Theta^m}$ for $m = \{1, 2, 5, 10\}$. The red line represents the step function, which is a binary discretization of the Hill function.

The dynamic behaviour of a probabilistic Boolean network can be analyzed using Markov chains. For a detailed definition, please refer to [19, 20].

In this report, we deal with the assembly and analysis of regular Boolean networks. The following section gives a brief introduction to the theory of Boolean networks.

1.2 Boolean networks

Modeling with Boolean logic is a widely-used method in systems biology [12]. Boolean networks can model the dynamics and the influence of one variable on another over time. This technique operates on discretized variables with two states (e.g. *high/low, yes/no, on/off, 0/1, ...*). In the following, we use the values $\{0, 1\} =: \mathbb{B}$. A variable in this Boolean space is called a Boolean variable. Let $\mathbf{x} = (x_1, \ldots . x_n)^T$, $x_i \in \mathbb{B}$, denote the vector of Boolean variables which are involved in a Boolean network. The changes of these variables over time are modeled by a set of functions $\Theta = \{f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$.

There are different ways to model time in this context. Here, synchronous time-discrete Boolean networks are used, which means that all variables are updated synchronously in the following way:

$$x_i(t+1) = f_i(\mathbf{x}(t)) \text{ for } i \in 1, \ldots, n. \tag{1}$$

These updates are called *transitions*. Prior to a more formal definition of such (Boolean) functions f, note that $f : \mathbb{B}^n \to \mathbb{B}$. Both input and output space of f are finite, and all possible input-output combinations can be summarized in a finite set

$$c_f = \{(\mathbf{x}, f(\mathbf{x})) | \mathbf{x} \in \mathbb{B}^n\} \tag{2}$$

The elements of c_f can be ordered in a so-called *truth table*. An example of such a table for a three-dimensional input space can be seen in Table 1. The first n columns contain the assignments to the input variables. The last column contains the vector of output values $\mathbf{y}$. The table consists of 2^n rows which correspond to the different input combinations. We assume that the table is filled in some canonical order. In our case, a Boolean table of n variables is sorted increasingly according to $dec_n : \mathbb{B}^n \to \{1, \ldots, 2^n\}$

$$dec_n(\mathbf{x}) = \sum_{i=1}^{n} 2^{i-1} x_i + 1 \tag{3}$$

applied to the input vectors. In this way, the input vector at position i can be seen as the binary encoding of $i - 1$. The inverse function is

$$bin_n(i) = dec_n^{-1}(i) \tag{4}$$

Two Boolean functions f_i, f_j can be distinguished according to their $\mathbf{y}$ vectors, $\mathbf{y}_i \neq \mathbf{y}_j$ for all $i \neq j$. If no further restrictions exist, the set of all Boolean functions over n variables consists of 2^{2^n} elements.

x_1	x_2	x_3	$\mathbf{y}$
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

Table 1: Truth table of a 3-dimensional function

Note that not all variables x_i must have an influence on a given function f. A variable that does not affect f is called fictitious [15].

Definition A Boolean variable x_j is called *fictitious* for a Boolean function f if

$$f\left((x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n)^T\right) = f\left((x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_n)^T\right) \tag{5}$$

Otherwise, it is called *non-fictitious*.

A truth table – which is an equivalent representation of a certain Boolean formula f – merely has to contain non-fictitious variables. In the following, we assume Boolean tables to be reduced in this respect. f can be reconstructed by knowing $\mathbf{y}$ and the non-fictitious variables.

We now introduce a more structural definition of Boolean formulas.

Definition Let $\mathbf{x} = (x_1, \ldots .x_n)^T$, $x_i \in \mathbb{B}$ be a vector of Boolean variables. A Boolean function is a function of the form $f : \mathbb{B}^n \rightarrow \mathbb{B}$. A *Boolean function* is either atomic $f(\mathbf{x}) = x_i$, $i \in \{1, \ldots, n\}$ or one of the three following compositions of Boolean formulas g and h:

$$f(\mathbf{x}) = g(\mathbf{x}) \wedge h(\mathbf{x}) \qquad \text{conjunction} \tag{6}$$

$$f(\mathbf{x}) = g(\mathbf{x}) \vee h(\mathbf{x}) \qquad \text{disjunction} \tag{7}$$

$$f(\mathbf{x}) = \neg g(\mathbf{x}) \qquad \text{negation} \tag{8}$$

Here, $\wedge$,$\vee$,$\neg$ are the basic Boolean functions AND, OR and NOT. Their truth tables can be found in Table 2. Although there are other well-known Boolean functions, such as XOR or implication, we focus on these three functions which suffice to construct any Boolean formula [21].

$\wedge$

x_1	x_2	$\mathbf{y}$
0	0	0
1	0	0
0	1	0
1	1	1

$\vee$

x_1	x_2	$\mathbf{y}$
0	0	0
1	0	1
0	1	1
1	1	1

$\neg$

x_1	$\mathbf{y}$
0	1
1	0

Table 2: The Boolean functions AND, OR and NOT

Definition A *Boolean network* is an ordered pair $N = (X, \Theta)$, where X is a non-empty finite set of Boolean variables $\{x_1, \ldots, x_n\}$, and Θ is a set of functions $\{f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$ with $\mathbf{x} = (x_1, \ldots, x_n)^T$.

An example Boolean network is shown in Fig. 3.

The functions in Θ describe how the values of the variables in X change over time. In the following, we only consider synchronous time-discrete updates

$$x_i(t + 1) = f_i(\mathbf{x}(t)) \text{ for } i \in 1, \ldots, n. \tag{9}$$

These updates are called transitions. For better reading, let $F_\Theta : \mathbb{B}^n \rightarrow \mathbb{B}^n$ be the function

$$F_\Theta(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))^T \tag{10}$$

for $\Theta = \{f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$. In this way, transitions can be written as

$$\mathbf{x}(t + 1) = F_\Theta(\mathbf{x}) \tag{11}$$

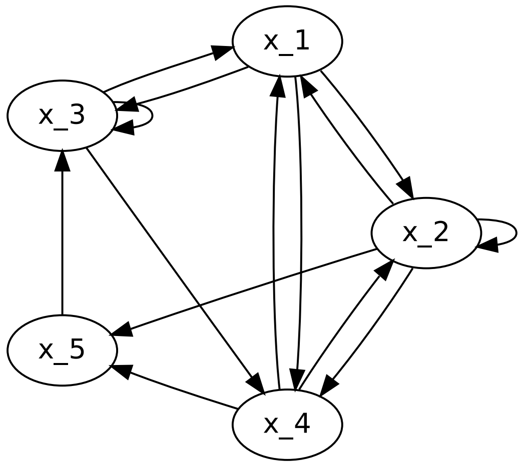

Figure 3: A Boolean network with 5 variables.

Definition A *directed graph* G is an ordered pair $G = (V, E)$, where V is a non-empty finite set of variables called *nodes*, and E is a set of ordered pairs of nodes (u, v), $u, v \in V$, written as $u \to v$ or $v \leftarrow u$. The elements of E are called *directed edges*.

Definition Let $G = (V, E)$ be a directed graph. A *path* in G is a sequence of nodes $u_1, \ldots, u_m$, $m \geq 1$ such that $(u_i, u_{i+1}) \in E$ for $1 \leq i < m$. The length of a path is the number of nodes.
A path $u_1, \ldots, u_m$ is *simple* if all nodes in the path are distinct.
A path $u_1, \ldots, u_m$, $m \geq 1$, is a *cycle* or *attractor* if $u_1 = u_m$.
A cycle of length one is also called *steady-state* or *singleton attractor*.

Definition Let $G = (V, E)$ be a directed graph. Let $u_1, \ldots, u_m = U \subset V$, $m \geq 1$, denote an attractor in G. The *basin* of U is the set of all nodes $v \in V$ for which a path from v to a node $u \in U$ exists.

Definition Let $N = (X, \Theta)$ be a Boolean network and $n = |X|$ be the number of variables in N. A *state graph* of N is a directed graph $G = (S, E)$ with nodes $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and directed edges

$$E = \{(\mathbf{x}_i, \mathbf{x}_j) \mid F_\Theta(\mathbf{x}_i) = \mathbf{x}_j,\, i, j \in \{1, \ldots, k\}\}\,, \tag{12}$$

where $\mathbf{x}_i \in \mathbb{B}^n$ and $k = 2^n$.

The state graph of the Boolean network in Fig. 3 is depicted in Fig. 4.

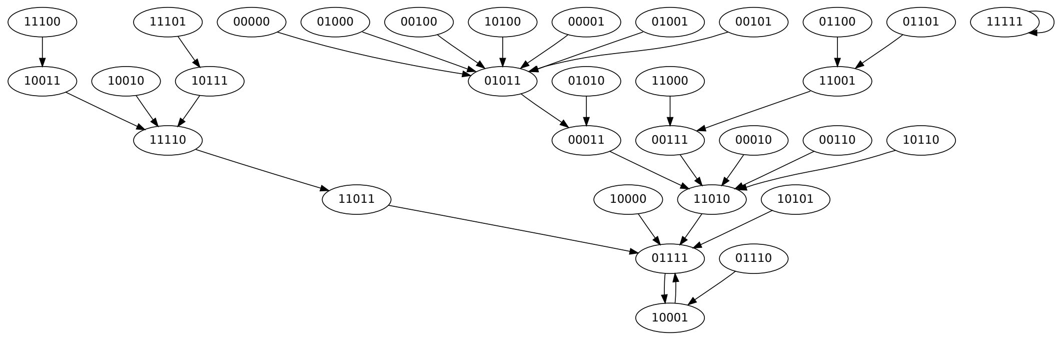

Figure 4: The state graph of the Boolean function in Fig. 3

The number of different states in a state graph corresponds to the number of different values a Boolean vector $\mathbf{x} \in \mathbb{B}^n$ can take. Note that $F_\Theta(\mathbf{x})$ is a deterministic function. As a consequence, each node has exactly one outgoing edge. As $F_\Theta : \mathbb{B}^n \to \mathbb{B}^n$, the output space consists of only 2^n elements. Therefore, each sequence of state transitions ends up in an attractor after at most 2^n steps, and at least one attractor exists.

1.3 Learning from examples

In a setting where predefined rules (Boolean functions) exist, the Boolean network can be constructed and analyzed by applying the theory above. If such a rule set is not available, the rules have to be inferred from a set of measurements, a so-called *dataset*, in a preprocessing step. The following is based on the theory of Lähdesmäki et al. [15]. A dataset $\mathcal{D}$ for a Boolean function f consists of a set of m triples

$$\mathcal{D} = \{(\mathbf{x}_1, l_1, w_1), \ldots, (\mathbf{x}_m, l_m, w_m)\}, \tag{13}$$

where $\mathbf{x}_i \in \mathbb{B}^n$ is an example input, and $l_i \in \mathbb{B}$ is the corresponding output, a label given to the example by an expert. The variables $w_i \in \mathbb{R}$ are optional weights which can be used to model the importance of a single example. If no information about the importance is known, w_i is set to 1 for all $i \in \{1, \ldots, m\}$.

Let us at first assume that the expert does not make any mistakes, all labels are correct, and no ambiguous examples exist. The examples can then be used to reconstruct the truth table, in particular $\mathbf{y}$. Assume that $\mathbf{y}$ is initially filled with *don't care* symbols, for example $y_i = *$ for all i. The reconstruction then looks as follows:

$$y_{\mathrm{dec_n}(\mathbf{x})} = l \quad \forall (\mathbf{x}, l, w) \in \mathcal{D} \tag{14}$$

It can easily be seen that a dataset of 2^n distinct examples is needed to reconstruct the complete truth table. Common datasets contain $m \ll 2^n$ examples. A truth table that comprises $*$ symbols does not correspond to a Boolean function. It represents a *partially defined Boolean function*.

Definition A *partially defined Boolean function* is a function of the form $pf : \mathbb{B}^n \rightarrow \{0, 1, *\}$.

Note that each Boolean function is a partially defined Boolean function. As long as $*$ symbols can be found in a truth table, it cannot be used to predict a Boolean value for each input. In the next step, we therefore replace a partially defined Boolean function by a Boolean function which at least predicts known values of the truth table correctly. We first define the set of positive and negative examples of a function.

Definition Let pf denote a partially defined Boolean function. The *set of known examples* of pf is defined as

$$\mathrm{tra}(pf) = \{(\mathbf{x} \mid \mathbf{x} \in \mathbb{B}^n, pf(\mathbf{x}) \neq *)\} \tag{15}$$

A Boolean function which fulfils the above condition is called a *consistent extension* .

Definition A *consistent extension* of a partially defined Boolean function pf is a Boolean function f for which

$$f(\mathbf{x}) = pf(\mathbf{x}) \quad \forall \mathbf{x} \in \mathrm{tra}(pf) \tag{16}$$

A candidate for a consistent extension has to replace all $*$ symbols of pf by a value of $\mathbb{B}$. If pf contains q $*$ symbols, there are consequently 2^q equally suited candidates in this setting. In the worst case, all $*$ symbols are set to the wrong Boolean value, which leads to q wrong entries in the truth table.

Usually, some prior knowledge or assumptions exist on how the final structure of the Boolean function should look like. For example, the indegree of the function may not exceed a certain value. The class of Boolean functions can be restricted to a smaller concept class $\mathcal{C}$ according to such constraints. There is no guarantee that function $f \in \mathcal{C}$ for such a restricted class exists that fulfils the criterion of a consistent extention. If no such function exists, we have to switch to the *best-fit extension* criterion.

Definition Let pf be a partially defined Boolean function and $\mathcal{C}$ a concept class. Furthermore, let $\mathcal{D} = \{(\mathbf{x}_1, l_1, w_1), \ldots, (\mathbf{x}_m, l_m, w_m)\}$ be the dataset which was used to determine pf. A *best-fit extension* of a partially defined Boolean function pf is a Boolean function $f \in \mathcal{C}$ with

$$f = \mathrm{argmin}_f \sum_{i=1}^{m} w_i |pf(\mathbf{x}_i) - f(\mathbf{x}_i)| \tag{17}$$

Note that by this formulation, two identical examples in $\mathcal{D}$ can be assigned two different weights.

So far, only datasets which do not include contradictory labels for examples have been considered. If there are contradictions, the corresponding vector $\mathbf{y}$ of a partially defined

Boolean function pf cannot be calculated as in Equation 14. For each y_i, the plausibility of a 0 value (c_i^0) or a 1 value (c_i^1) has to be determined:

$$\mathbf{c}_i^0 = \sum_{(\mathbf{x},l,w)\in D} w\mathbb{I}_{[\deg(\mathbf{x})=i,l=0]} \tag{18}$$

$$\mathbf{c}_i^1 = \sum_{(\mathbf{x},l,w)\in D} w\mathbb{I}_{[\deg(\mathbf{x})=i,l=1]} \tag{19}$$

The entries of $\mathbf{y}$ are then set to

$$y_i = \begin{cases} 0 & \text{if } c_i^0 > c_i^1 \\ 1 & \text{if } c_i^0 < c_i^1 \\ * & \text{if } c_i^0 = c_i^1 \end{cases} \quad \text{forall } i \in \{1,\ldots,2^n\} \tag{20}$$

The final vector $\mathbf{y}$ again defines the partially defined Boolean function pf.

The training of a Boolean network can be seen as the training of all its functions. Assume that we have a set of measurements $\{(\mathbf{x_1(t)}, \mathbf{x_1(t+1)}), \ldots, (\mathbf{x_m(t)}, \mathbf{x_m(t+1)})\}$ with $\mathbf{x_m}(t) = (x_{m1}(t), \ldots, x_{mn}(t))^T$. An example d_j of the training set $\mathcal{D}_i$ of function f_i is then

$$d_j = \big((x_{j1}(t), \ldots, x_{jn}(t))^T, x_{j,i}(t+1), w\big) \tag{21}$$

The above training procedure can now be applied to such examples.

1.4 Fields of interest

1.4.1 Integration of literature knowledge into Boolean networks

The assembly of Boolean gene-regulatory networks does not necessarily require direct output from experiments. Rules can also be compiled from abstract conclusions of previous research. A typical way of building or extending a Boolean network from literature comprises the following steps:

1. At first, the particular subject of research must be identified. This includes determining the most important genes involved in the process as well as the expected characteristics or phenotypes one wants to distinguish.

2. Expression patterns of the involved genes must be available to constitute the expected outcome of simulations.

3. Now, interaction patterns can be extracted from literature and transferred into Boolean rules.

4. In a simulation step, the found rules are applied to binary input data, and the results can be compared to the expected expression patterns. As an input, one can either test all combinations of gene values (*on* or *off* for each gene), or a set of predefined "meaningful" inputs can be tested.

5. In the process of biological model validation [10], the simulation results are compared to the expected expression patterns. If the simulated results do not match these expression patterns, conclusions can be drawn. This may result in the insertion of new – previously unknown – rules inferred from the resulting patterns. Then, further simulation steps follow to validate the new network.

Here, the Boolean rules are described in a fixed format according to the following grammar (in EBNF):

```
Rule = GeneName ", " BooleanExpression;
BooleanExpression = GeneName
                  | "!" BooleanExpression
                  | "(" BooleanExpression ")"
                  | BooleanExpression " & " BooleanExpression
                  | BooleanExpression " | " BooleanExpression;
GeneName = ?A gene name from the list of involved genes?;
```

where `!` stands for a negation, `&` denotes a conjunction of two Boolean expressions, and `|` denotes a disjunction of two expressions.

1.5 Finding and analyzing attractors in Boolean networks

As we have already seen in Section 1.2, attractors form stable cycles of states in Boolean networks. States that are not included in an attractor are only visited along paths to an attractor. Thus, attractors comprise the states in which a gene-regulatory network resides most of the time. This means that attractors carry strong biological implications, and the identification of interesting attractors is of great interest to researchers [14].

A simple procedure to identify all possible attractors of a Boolean network $N(X, \Theta)$ is the following:

- A set of possible input value combinations for the genes in X is defined. By default, all possible 2^n input value combinations for n genes are taken, but one can also restrict the input values to those values that are biologically plausible.
- For each input value vector, set the current network state $x(0)$ to the vector and perform the following steps:
  1. Set the iteration counter $t = 0$, and mark $x(0)$ with iteration 0.
  2. Apply F_Θ to the current state $x(t)$, resulting in a new state $x(t+1)$.
  3. If the state $x(t+1)$ has not yet been visited, mark $x(t+1)$ with the current iteration t, set $t = t + 1$, and go to step 2.
  4. If the state $x(t+1)$ has already been visited and is marked with iteration k, we have found an attractor of length $t - k + 1$.

This algorithm is exponential in the number of input genes. Although there exist more sophisticated algorithms with improved complexity, these algorithms are still exponential in the number of input genes. This is due to the fact that the problem of finding steady-state attractors in Boolean networks has been shown to be NP-hard [23].

As mentioned previously, each Boolean network has at least one attractor. Most networks, however, include a number of attractors, of which not all may be biologically meaningful. It is thus important to define measures that help selecting those attractors that are of interest from a biological point of view.

According to Kauffman [14], changes of attractors and their basins due to mutations in the Boolean rules are of immediate interest in this respect. He proposes to investigate the stability of attractors to such perturbations.

We present two ways of measuring the importance of an attractor. Both are based on random structural perturbations of the networks. With this method, we compare the original network to a set of K randomly tampered copies of the network. We employ two methods of perturbing the original rule set:

- **Random bitflips**: From the truth table result column $\mathbf{y}_i$ of a randomly chosen gene i, a random number of bits is negated. This leads to a change of the function F_Θ in one component.
- **Random shuffle**: The truth table result column $\mathbf{y}_i$ of a randomly chosen gene i is permuted randomly. This again changes the function F_Θ in one component. We propose this method because it preserves the number of ones and zeros in the truth table.

Random bitflips have been previously employed to perturb Boolean networks [9, 22]. The effects of random bitflips on the structure of Boolean networks have been investigated in these papers. It was shown in [9] that redundant nodes can increase the robustness against such perturbations.

We create K copies for each of the tampering methods and then measure two factors to assess the importance of an attractor:

Number of occurrences of the attractor in randomly changed networks: This importance measure is based on the assumption that, when tampering a network randomly, a biologically meaningful attractor should still be identified in the majority of changed networks, as such an attractor should be robust against small mismeasurements. Counting the number of occurrences of the originally identified attractor in the K tampered copies provides information on the robustness and stability of the attractor.

Comparison of sizes of the attractor basins: A biologically meaningful attractor is likely to be reachable from a high percentage of the biologically plausible states. A good measure is thus the number of biologically plausible states – i.e. the states that probably occur in real gene-regulatory networks – in the basin of an attractor. Yet, it is usually not known in detail which of the states are biologically plausible. To approximate this measure, one can simply take the size of the complete attractor basin as a measure of importance of the attractor.

If the original network comprises a true attractor, the basin size of this attractor should be extreme in comparison to randomly changed networks. In our evaluations, we count the number of tampered rule sets in which the basin size is smaller, greater or equal to the original network for each of the attractors. To be biologically relevant, we expect the basin size of a perturbed attractor to be smaller than or equal to the original attractor.

Computer-intensive testing: Values returned by the above measures are not easy to interpret, as they lack comparable values from "bad" attractors. This can be overcome by computer-intensive tests: For each tested network, we create a set of R networks with the same number of genes, but with the transition functions F_Θ consisting of truth tables that were drawn randomly. For all these networks including the original network, K tampered copies are created, and the above importance measures are applied. If the attractors of the original network are meaningful regarding these importance measures, one would expect that their values for the importance measures are extreme compared to the values of randomly created networks. For a significance test with level α (usually $\alpha = 0.05$), we call an importance measure value significant if it is greater or equal than the $1 - \alpha$ quantile of the distribution of the importance measure over all random networks. For each of the tests, we calculate the following values from the sample of R random networks:

Test statistic: An estimate of the probability that the importance measure (i.e. the number of occurrences or the number of basins with a smaller or equal size within the K perturbed copies) of an attractor in a randomly created network is greater than the corresponding value of a certain attractor in a biological network.

95% confidence interval: The interval in which the true parameter for the above statistic in the population of **all** random networks is located with a probability of 95%.

p-**value:** The percentage of attractors in random networks that achieve a higher importance measure value than the chosen attractor of the biological network.

Analysis of perturbation methods: One basic assumption of the above testing procedure is that the perturbation methods disrupt little of the structure of the original network, such that a perturbed biological network is somehow still "less random" than a network that was generated completely randomly. In order to analyze the effects of the perturbation operators in detail, we perform additional experiments: Again, K randomly changed copies of a biological network are created for each of the perturbation methods. We now measure the normalized Hamming distances of the original network and each of its perturbed copies as proposed in [9]:

$$dS = \frac{1}{|c_f|} \sum_{\mathbf{x} \in c_f} H(F_O(\mathbf{x}), F_C(\mathbf{x})), \tag{22}$$

where $F_O(\mathbf{x})$ is the transition function of the original network O for the initial state $\mathbf{x}$ taken from the set of all states c_f, $F_C(\mathbf{x})$ is the transition function of the perturbed copy C for the initial state $\mathbf{x}$, and $H(\mathbf{x}, \mathbf{y})$ is the Hamming distance of the vectors $\mathbf{x}$ and $\mathbf{y}$, i.e. the number of components in which the vectors differ.

We take the mean value of all K dS values to measure the average disruption of a network by a perturbation method. Note that the worst-case value of dS – when the two state spaces are not correlated – is 0.5; if the two networks do not differ, it is 0 (see [9]).

2 Simulations

We assess the robustness of attractors in three example networks from literature using the methods described in the previous section: We first apply random perturbations to the networks to determine the stability of the attractors. We then compare the resulting robustness measures to the corresponding measures on a set of randomly generated networks in computer-intensive tests. For each of the networks, we also analyze the effects of the perturbation methods on the network structure.

2.1 Networks

2.1.1 The mammalian cell cycle

The mammalian cell cycle consists of four phases. We employ a simplified Boolean network model for the cell cycle with 10 genes developed by Fauré et al. [7]. The network has one attractor of length 1 and one attractor of length 7. It is depicted in Fig. 5.

2.1.2 Segment network of Drosophila melanogaster

This network describes expression patterns of the segment polarity genes of the fruit fly Drosophila melanogaster. The genes are expressed in stages of embryonical development to construct the segments of the fruit fly. The Boolean network originally consists of 15 genes and was proposed by Albert et al. in 2003 [1]. It has 10 steady-state attractors. We employ a version of the network in which several variables have been eliminated, resulting in a network with 8 genes, of which 4 are set to a fixed value (see Fig. 6, [22], and Eq. 4 of [1]).

2.1.3 The yeast cell cycle

The cell cycle of budding yeast consists of four phases, in which a total of around 800 genes are involved. Li et al. [16] present a reduced network with 11 genes. In total, there are 7 attractors of length 1 (steady-state attractors), but with extremely different sizes of the basins. The attractor with the largest basin (1764 states) was described by the authors as being relevant for the first phase of the cell cycle, the G_1 state. The network was originally designed as a logical network, in which a state can be dependent on more

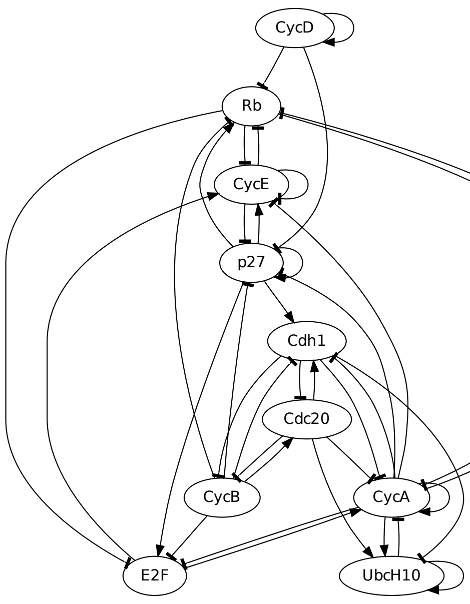

Figure 5: The mammalian cell cycle network (Fauré et al., 2006)

than one previous time step. This property was used by the authors to describe a self-regulation step after *td* time steps. We set this constant to 1 to convert the network to a Boolean net. The network is shown in Fig. 7.

2.2 Results

2.2.1 Random perturbations of biological networks

For each of the described networks, 1000 randomly changed copies were created using random bitflips, and 1000 copies were created using random shuffle. The attractors from the original network were searched in the copies, and the occurrences were counted. If the attractor existed in a copy, the size of the attractor basin was determined and compared to the size of the basin in the original network. The results were divided into the categories *smaller basin than the original basin, equally-sized basin* and *larger basin*. Results are depicted in Table 3. A robust attractor should not be extinguished by small random changes of the network; hence, the fifth column of the table, which counts the number of occurrences of the original attractor in the randomly perturbed networks, is the primary indicator of robustness of an attractor. In addition, we expect a good attractor to have a larger basin than a randomly changed copy of this attractor on average, thus such an attractor should maximize the sum of the second and the third column of the table (basin size is less or equal in randomly changed networks).

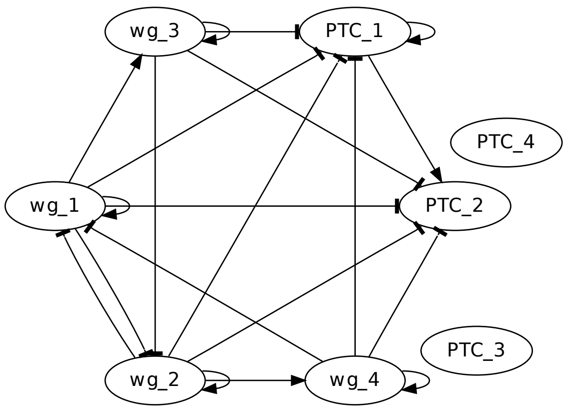

Figure 6: The simplified Drosophila melanogaster segment network (Albert et al., 2003)

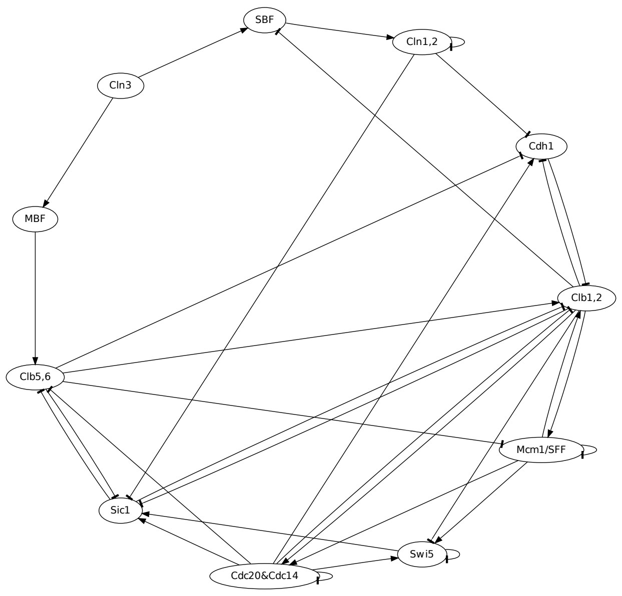

Figure 7: The yeast cell cycle network (Li et al., 2004)

In the Mammalian cell cycle network, we see that the large attractor with 7 genes (attractor 2) is much more susceptible to small changes in the network than the smaller attractor (attractor 1): In the randomly perturbed networks, attractor 2 occurs only half as often as attractor 1. In these occurrences, the number of enlarged basins is much higher than the number of decreased basin sizes for the large attractor, whereas the number of smaller basin sizes in the small steady-state attractor is 4 times as high as the number of larger basins for the random bitflip tests. Randomly shuffled copies generally lead to lower number of occurrences than random bitflips. Surprisingly, there is no occurrence with a larger basin in any randomly shuffled copy for both attractors.

The number of occurrences of all 10 steady-state attractors in random copies of Drosophila melanogaster network is similar, both for random bitflips and random shuffles. An exception is attractor 4, which has a significantly higher number of occurrences than the other attractors in the random shuffling experiments. Again, the number of occurrences for random shuffling is much lower than the number of occurrences for random bitflips. The distributions of the basin sizes differ among the attractors: While randomly tampered copies of the networks mostly expose smaller or equally-sized basins for attractors 2, 3, 4, 7 and 9, the basins grow on average for attractors 5, 6, 8 and 10. This behaviour is consistent for both perturbation methods. Attractor 1 does not show a consistent behaviour. All attractors have a comparatively high number of occurrences in which the basin size stays the same between the original network and the randomly changed networks.

The attractors of the yeast cell cycle network have very high numbers of occurrences in the perturbed networks. Attractor 2 is the attractor which was identified as being biologically relevant by the authors of the network. This attractor occurs in 952 of the randomly changed networks. Its relevance is also supported by the very high number of randomly changed networks in which the basin size is lower than in the original networks. Attractor 1 is even found in all 1000 random copies of both perturbation methods. The basin size grows smaller in the randomly changed networks on average, but the number of changed networks with a smaller basin size is much lower than that of attractor 2. Attractors 1 and 2 appear to be the most robust attractors in this network. Other attractors that show smaller basin sizes on average in the randomly perturbed networks are attractors 3, 4 and 7. The size of the basin does not seem to change much for attractor 6, and the behaviour is slightly different for the two perturbation methods. Attractor 5 has a greater or equal basin size in all random copies of both methods, indicating a low robustness and thus possibly a reduced biological importance of the attractor.

2.2.2 Computer-intensive tests

To assess the relevance of the attractors of the three biological networks in comparison to randomly generated networks, we perform computer-intensive tests as described in Section 1.5. For each test, $R = 1000$ randomly generated networks with the same size as the corresponding original networks are modified randomly $K = 1000$ times, the relevance measure is calculated for each of the 1000 networks, and the resulting distribution is compared to the relevance measure of the real biological network. As relevance measures, both the number of occurrences of the original attractor in the modified copies and the number of times the basin of the modified attractor is smaller than or equal to

	Random bitflips					Random shuffle				
Attr. no.	Attractor length	Basin bio-net larger	Basins equal	Basin bio-net less	# found in copies	Attractor length	Basin bio-net larger	Basins equal	Basin bio-net less	# found in copies
	Mammalian cell cycle					Mammalian cell cycle				
1	1	222	455	50	727	1	228	327	0	555
2	7	5	284	47	336	7	5	259	0	264
	Drosophila melanogaster					Drosophila melanogaster				
1	1	53	606	98	757	1	85	435	0	520
2	1	148	528	73	749	1	108	328	88	524
3	1	112	556	63	731	1	103	318	90	511
4	1	213	370	163	746	1	130	400	237	767
5	1	18	578	165	761	1	33	411	192	636
6	1	19	572	148	739	1	39	386	218	643
7	1	120	556	63	739	1	118	318	23	459
8	1	19	624	99	742	1	39	323	214	576
9	1	141	528	69	738	1	115	328	21	464
10	1	18	638	95	751	1	33	331	230	594
	Yeast cell cycle					Yeast cell cycle				
1	1	240	656	104	1000	1	265	642	93	1000
2 (*)	1	822	101	29	952	1	588	317	47	952
3	1	260	516	153	929	1	270	473	177	920
4	1	527	170	263	960	1	477	317	199	993
5	1	0	877	98	975	1	0	824	133	957
6	1	232	495	210	937	1	246	454	261	961
7	1	412	270	202	884	1	426	351	156	933

Table 3: Number of occurrences and changes of sizes of the attractor basins for different networks in comparison to 1000 randomly perturbed copies of the networks. The first column is a running number to identify the attractors of each dataset. In the two following 5-column blocks, the first column lists the number of genes in the attractor, the second column contains the number of copies with a smaller basin for this attractor, the third column contains the number of copies with an equally-sized basin, the fourth column lists the number of copies with a larger basin, and the fifth column contains the overall number of copies in which the attractor existed. Each row describes a single attractor of the corresponding network.

the basin of the original attractor are taken. Each of the tests is conducted with random bitflips and with random shuffling. Thus, for each datasets, there exist 2 tests based on the number of occurrences and 2 tests based on the basin size with different perturbation methods.
To get a better understanding of the behaviour of the tests for different perturbation methods, we also analyze the effects of these methods on the underlying biological networks as described in Sec. 1.5.

On the mammalian cell cycle network, the tests with the numbers of occurrences – the simpler measure – are all far from being significant (see Fig. 8). The numbers of occurrences in copies of the biological network are sometimes even less than the mean value of the corresponding distributions of occurrences randomly generated networks with 10 genes, especially when applying random bitflips. However, in the basin size test, which employs the more complex measure, the steady-state attractor is significant with a p-value of 0.036, and the attractor of length 7 is close to significance with a p-value of

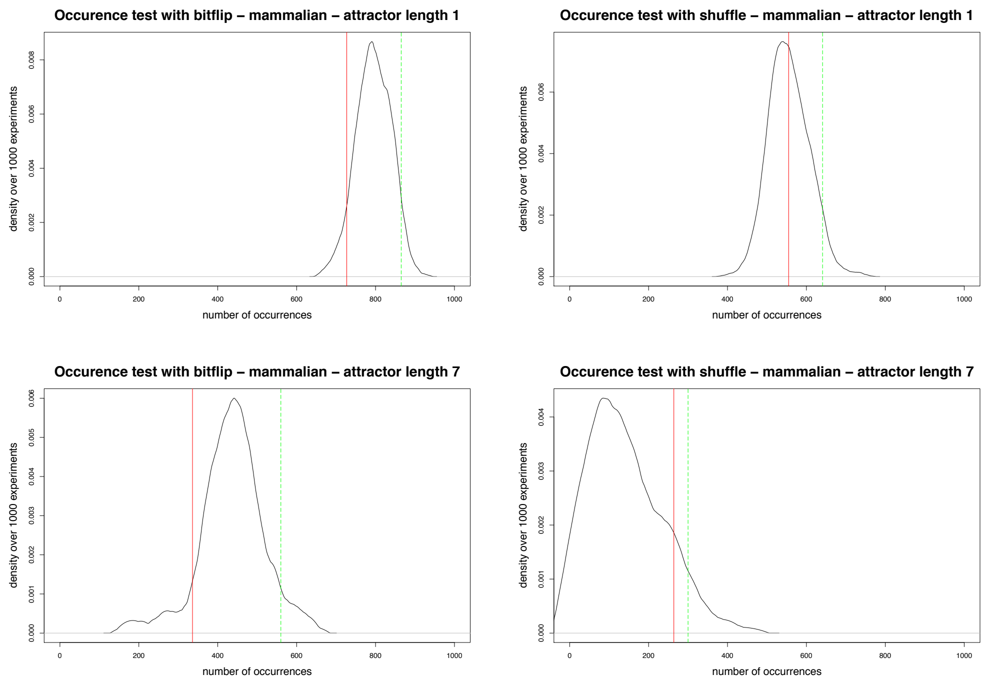

Figure 8: Distributions of the number of occurrences of attractors with 1 gene and with 7 genes in 1000 randomly perturbed copies of 1000 random networks with 10 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the mammalian cell cycle network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0.937	0.921–0.952	0.94
2	0.925	0.871–0.978	0.925
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0.479	0.448–0.51	0.481
2	0.097	0.037–0.157	0.097

0.065 when applying random shuffling (see Fig. 9) . Again, p-values for random bitflips are much greater.

The higher significance of the random shuffling tests compared to random bitflips can be explained by the analysis of the perturbation methods: On the mammalian cell cycle network, random bitflips seem to completely destroy the network structure, leading to an average Hamming distance to the original network of 0.496 (close to the worst-case

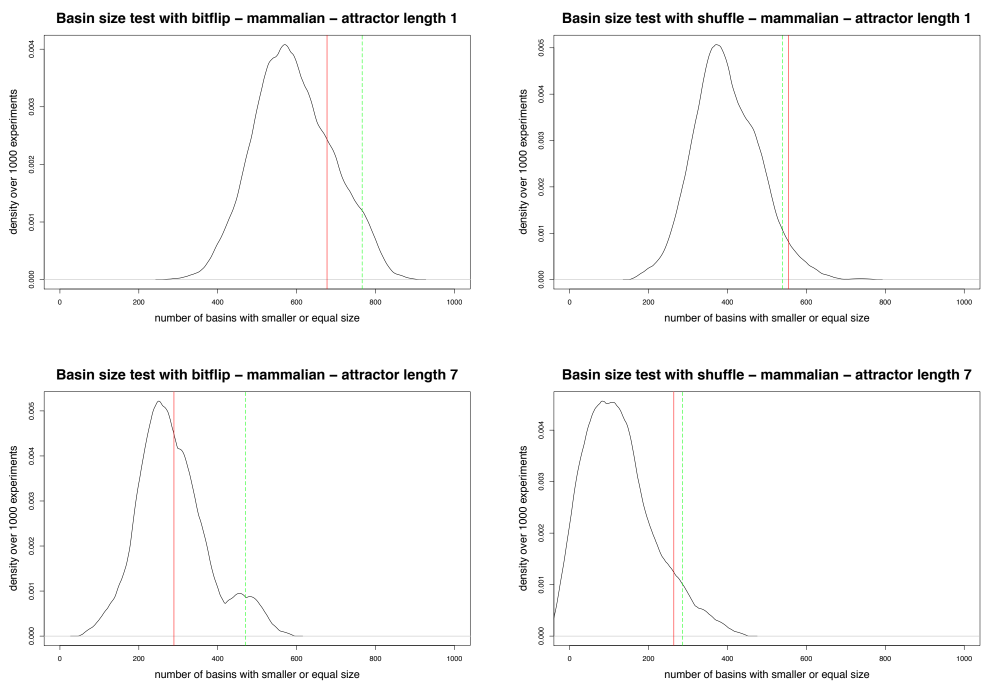

Figure 9: Distributions of the number of attractor basins that were smaller than or equal to the basins of the corresponding original attractors with 1 gene and with 7 genes in 1000 randomly perturbed copies of 1000 random networks with 10 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the mammalian cell cycle network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0.2	0.175–0.225	0.202
2	0.43	0.329–0.531	0.43
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0.036	0.024–0.047	**0.037***
2	0.065	0.015–0.114	0.065

value of 0.5). Random shuffling, however, leads to a much smaller average distance of 0.356.

On the Drosophila melanogaster network, one of the 10 steady-state attractors – attractor 4 – is highly significant in the occurrence test with random shuffling. This corresponds to the results in Table 3, where this attractor occurs far more often in randomly shuffled networks than all other attractors. In the random bitflip test, the

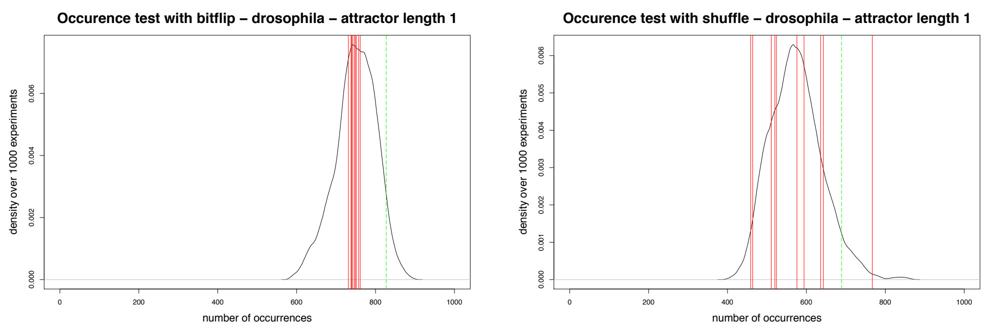

Figure 10: Distributions of the number of occurrences of attractors with 1 gene in 1000 randomly perturbed copies of 1000 random networks with 15 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the Drosophila melanogaster network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0.463	0.433–0.494	0.468
2	0.524	0.493–0.555	0.538
3	0.663	0.634–0.693	0.67
4	0.553	0.523–0.584	0.555
5	0.43	0.399–0.46	0.441
6	0.594	0.564–0.624	0.604
7	0.594	0.564–0.624	0.604
8	0.57	0.54–0.601	0.577
9	0.604	0.574–0.634	0.615
10	0.514	0.483–0.545	0.519
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0.777	0.752–0.803	0.783
2	0.764	0.738–0.791	0.769
3	0.822	0.798–0.845	0.827
4	0.005	0.001–0.009	**0.005***
5	0.163	0.141–0.186	0.165
6	0.147	0.125–0.168	0.148
7	0.979	0.97–0.988	0.979
8	0.458	0.428–0.489	0.462
9	0.975	0.966–0.985	0.976
10	0.357	0.328–0.387	0.361

occurrence values of the 10 attractors are close to the mean value of the randomly generated networks with 15 genes (see Fig. 10). As on the mammalian cell cycle network, it seems that for the Drosophila network, random bitflips destroy the structure of the network, making it similar to a randomly generated network. This is again confirmed

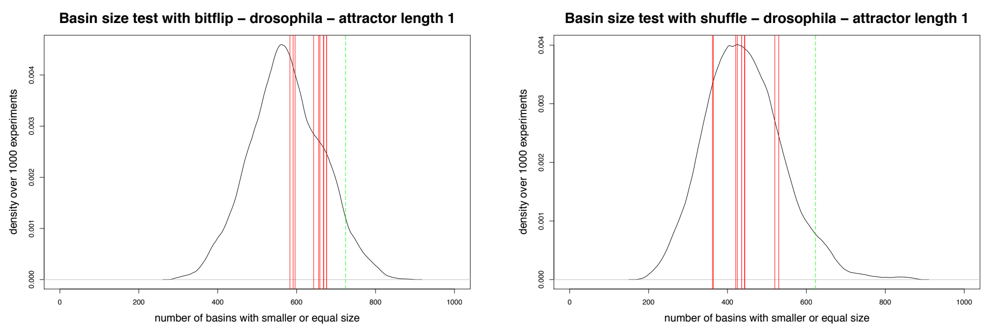

Figure 11: Distributions of the number of attractor basins that were smaller than or equal to the basins of the corresponding original attractors with 1 gene in 1000 randomly perturbed copies of 1000 random networks with 15 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the Drosophila melanogaster network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0.191	0.167–0.215	0.192
2	0.143	0.121–0.164	0.144
3	0.171	0.148–0.195	0.175
4	0.431	0.4–0.461	0.439
5	0.383	0.353–0.413	0.389
6	0.403	0.373–0.433	0.409
7	0.143	0.121–0.164	0.144
8	0.227	0.201–0.253	0.23
9	0.166	0.143–0.189	0.171
10	0.197	0.172–0.222	0.2
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0.196	0.172–0.221	0.199
2	0.492	0.461–0.523	0.499
3	0.562	0.532–0.593	0.565
4	0.174	0.151–0.198	0.177
5	0.461	0.431–0.492	0.465
6	0.546	0.515–0.576	0.55
7	0.492	0.461–0.523	0.499
8	0.789	0.764–0.814	0.795
9	0.465	0.435–0.496	0.469
10	0.777	0.752–0.803	0.783

by the Hamming distance measurements: For random bitflips, the average Hamming distance between the Drosophila network and the perturbed copies is 0.447, while the corresponding value for random shuffling is 0.298. The basin size tests resulted in no

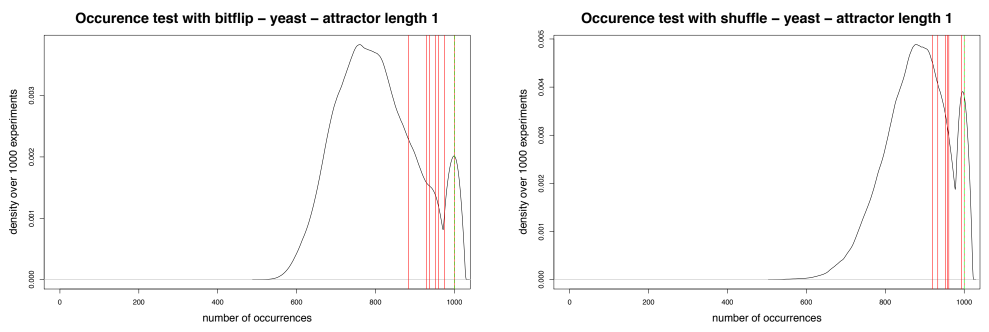

Figure 12: Distributions of the number of occurrences of attractors with 1 gene in 1000 randomly perturbed copies of 1000 random networks with 11 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the yeast cell cycle network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0	0–0	0.075
2	0.106	0.101–0.11	0.107
3	0.138	0.133–0.143	0.139
4	0.091	0.087–0.095	0.093
5	0.077	0.073–0.081	0.077
6	0.128	0.123–0.132	0.129
7	0.223	0.218–0.229	0.226
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0	0–0	0.103
2	0.194	0.188–0.199	0.196
3	0.321	0.315–0.328	0.325
4	0.104	0.1–0.108	0.104
5	0.175	0.169–0.18	0.179
6	0.162	0.157–0.167	0.164
7	0.265	0.259–0.271	0.268

significant attractors on this dataset (see Fig. 11). This may be explained by the fact that all attractors have a relatively high number of occurrences in which the basin size is equal to the randomly perturbed networks, which was discussed in Sec. 2.2.1.

In the tests with 11 genes corresponding to the yeast cell cycle network, several attractors have a p-value below 0.1 in the random bitflip experiments, both in the occurrence tests and in the basin size tests (see Figures 12 and 13). In particular, attractor 1, which we previously identified as one of the robust attractor, has p-values close to significance. Yet, the only significant result is the basin size test for the second robust attractor we

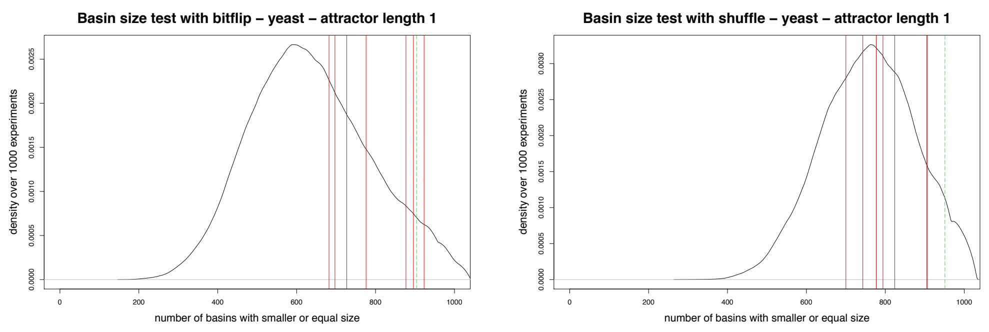

Figure 13: Distributions of the number of attractor basins that were smaller than or equal to the basins of the corresponding original attractors with 1 gene in 1000 randomly perturbed copies of 1000 random networks with 11 genes. On the left side, the copies were tampered using random bitflips, and on the right side, random shuffling was employed. The red lines correspond to the numbers of occurences of attractors in 1000 randomly perturbed copies of the yeast cell cycle network, which has the same number of genes. The green dashed line is the 95% quantile of the distribution.
The following table lists the test statistics with their corresponding confidence intervals and p-values for each attractor.

Random bitflip			
Attractor	statistic	conf. int.	p-value
1	0.056	0.052–0.059	0.057
2	0.038	0.035–0.041	**0.039***
3	0.182	0.177–0.188	0.184
4	0.324	0.317–0.33	0.326
5	0.071	0.068–0.075	0.072
6	0.264	0.258–0.271	0.267
7	0.357	0.351–0.364	0.359
Random shuffle			
Attractor	statistic	conf. int.	p-value
1	0.111	0.107–0.115	0.112
2	0.113	0.109–0.118	0.115
3	0.552	0.545–0.559	0.555
4	0.386	0.379–0.392	0.389
5	0.297	0.29–0.303	0.299
6	0.681	0.674–0.687	0.684
7	0.44	0.433–0.447	0.444

identified, attractor 2. This is the attractor that was identified by Li et al. [16] as being relevant for the G_1 state. The occurrence tests all fail, which is mainly due to the fact that the number of occurrences of attractors in the random networks is mostly quite close to the optimum (1000).

The results of the analysis of the perturbation methods is not as clear as for the other networks. Still, random shuffling achieves a slightly lower average Hamming distance of 0.390 compared to a value of 0.463 for random bitflips. Both method seem to affect the network structure significantly on this network.

3 Discussion

This report introduces the workflow of modeling gene-regulatory networks as Boolean networks with literature knowledge and analyzing their structure. In particular, we present methods to identify attractors and analyze their robustness and possible biological importance.

Extracting rule sets from literature constitutes an alternative to conducting costly experiments. Boolean networks are particularly suited for these design problems, as they do not require any quantitative information. It suffices to obtain qualitative statements about genes influencing each other.

In the process of designing a Boolean network from literature, determining attractors in a Boolean network can help to validate the network: The attractors should match known expression patterns of the biological network, as they constitute the states in which the network resides most of the time. If the attractors of the designed network do not match the expectations, it is even possible to infer previously unknown interactions by drawing conclusions from these results. In the appendix, we present an R package that finds and visualizes attractors in Boolean networks and is well suited for this validation and inference process.

We outlined that comparing the attractors in the original biological networks and a set of randomly modified networks can help to assess their relevance. In order to get reliable test results, a perturbation method must not change the structure of a network too drastically. Otherwise, all resulting networks are more or less random, which means that a test cannot distinguish between "real" random networks and networks derived from meaningful biological networks. Especially the random bitflip tests on the Drosophila network and the Mammalian cell cycle network seem to suffer from this problem. This is also confirmed by our analysis of the perturbation methods which measures the Hamming distance between the states of the original network and the randomly perturbed networks. This analysis suggests that random bitflips mostly disrupt the structure of a network stronger than the shuffling method we proposed. Yet, this may not be a general statement, as the random bitflip tests on the yeast cell cycle network are the only experiments with significant p-values on this network.

We employed two relevance measures for the attractors: The number of occurrences of an attractor in perturbed copies of the network, and the number of occurrences with a basin of smaller or equal size. The former simply checks whether the attractor itself is robust against small changes of the network, while the latter is more strict: In addition to existence of the attractor in the tampered networks, it assesses the size of its basin. The assumption is that the basin in the real biological network should mostly be larger than in a randomly modified copy. The results of our tests show that sometimes the occurrence measure is not sufficient to distinguish between random and non-random networks, and thus the basin size measure is the better choice for the tests. On the Drosophila network, however, results were clearer when applying the occurrence measure. A possible explanation for this is the fact that the basin sizes in this network often do not change when perturbing the network.

In all networks, tests were able to identify significant attractors. In the yeast cell cycle networks, an attractor that has previously been identified as being of biological relevance was significant in a test. However, our results also indicate that it is hard to define a single method that is able to reliably determine relevant attractors in networks of different structure and size: The significant results were produced by different configurations of perturbation methods and test statistics.

Acknowledgements: This work is supported by the German Science Foundation (SFB 518, Project C5), the Stifterverband für die Deutsche Wissenschaft (HAK), the Graduate School of Molecular Medicine at Ulm University funded by the Excellence Initiative of the German government (GSC270) and the Graduate School of Mathematical Analysis of Evolution, Information and Complexity at the University of Ulm (CM, HAK).

Appendix

The booleanAttractor package

booleanAttractor (available upon request) is an R package [18] designed to identify attractors in Boolean networks and to provide a visualization for them. It can read in networks in the format described in Section 1.4.1 and find attractors by trying all 2^n possible combinations of n variables. It is also possible to restrict the tried values to biologically plausible ones. The package runs on many platforms, in particular on Windows, Linux, and Macintosh. Due to the integration of the algorithms into the statistical platform R, it is easy to perform further tests and calculations on the resulting data and to combine the package with a broad variety of other biostatistical packages, for example from the Bioconductor project [8].

The following provides a short documentation of the functions in the package.

getattractor function

Retrieves attractors in the supplied network.

Usage:

```
getattractor(netinfo, filename, genesON = c(), genesOFF = c())
```

Arguments:

netinfo A net information structure returned by `readin.boolean()`

filename The name of a temporary file to store the identified attractors

genesON A vector of names of genes whose initial state is fixed to 1/ON to simplify the truth table

genesOFF A vector of names of genes whose initial state is fixed to 0/OFF to simplify the truth table

Details:

This function identifies attractors in a Boolean network according to the algorithm in Section 1.5. A network with n variables consists of truth tables of size 2^n, hence the complexity of the calculation is exponential in the input size. External C code is called to speed up calculations.

Value:

A list with components

attractor A list describing all found attractors. It is made up by the following sub-components:

- **aidx** The ranges of the vector *attractor* that belong to a specific attractor: The ith attractor is made up by the elements `attractor[(aidx[i] + 1):(aidx[i + 1])]`
- **attractor** An array of indices of truth table entries belonging to attractors. Must be split up according to *aidx*.
- **freq** The sizes of the attractor basins
- **numattractor** The total number of attractors found in the network

table A matrix with 3 columns that describes the attractor basins. The first column lists the indices of the states/rows in the truth table. The second column contains the number of transitions between the state and the corresponding attractor, and the third column is the index of this attractor in the *attractor* component.

Example

```
data(examplenet)
attractor = getattractor(net,"./attractor")
plotattractor2fig(attractor$attractor,net)
```

readin.boolean function

Reads in a Boolean network in the format specified in Section 1.4.1.

Usage:

```
readin.boolean(funcfile, seperate = ",")
```

Arguments:

funcfile The name of the file to be read

seperate The character used to separate the target variable from the formula. Defaults to ",".

Details:

Refer to the EBNF grammar in Section 1.4.1 for details on the file format. The rules are read in and converted to a truth table representation.

Value:

An internal network representation with the following structure:

interactions A list with n elements (for n genes), each describing the transition function for one gene. Each element is a list with two sub-elements:

- **input** A vector of k indices specifying the input genes for the function
- **func** The result column of the truth table for the k input genes with 2^k entries

genes A vector of n gene names

input, inputIndex, func, funcIndex Encoded vectors containing the same information as *interactions* that are used for easier information exchange with the C code.

fixed A vector that allows for fixing genes to biologically meaningful values. It contains an element for each of the n genes. 0 means the corresponding gene is fixed to 0/OFF, 1 means the corresponding gene is fixed to 1/ON, and -1 means the gene value is not fixed.

plotattractor2fig function

Plots a state table of one or several attractors.

Usage:

```
plotattractor2fig(attractorinfo, netinfo, sepinfo = list(), title = "",
plotfix = T)
```

Arguments:

attractorinfo The *$attractor* component of a return value of `getattractor()`, i.e. the attractor(s) to be plotted

netinfo A net information structure, usually the result of a call to `readin.boolean()`

sepinfo An optional structure to form groups of genes in the plot. This is a list with the following elements:

- **class** A vector of names for the groups. These names will be printed in the region belonging to the group in the plot.

index A list with the same length as *class*. Each element is a vector of gene indices belonging to the group.

title A string printed as the title of the plot

plotfix A logical value that determines whether genes with fixed values are included in the plot

Details:

For each attractor in *attractorinfo*, a figure is plotted to the currently selected device. This figure is a table with the genes in the rows and the states of the attractor in the columns. Cells of the table are red for 0/OFF values and green for 1/ON values. If *sepinfo* is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

Value:

This function has no return value.

Example:

```
data(examplenet)
attractor = getattractor(net,"./attractor")
plotattractor2fig(attractor$attractor,net)
```

plotattractor2tex function

Creates a LaTeX document with a state table of one or several attractors.

Usage:

```
plotattractor2tex(attractorinfo, netinfo, col = c(), sepinfo = list(), outfile
= "tex", title = "")
```

Arguments:

attractorinfo The *$attractor* component of a return value of `getattractor()`, i.e. the attractor(s) to be plotted

netinfo A net information structure, usually the result of a call to `readin.boolean()`

col An optional vector of two color name strings representing the 1/ON and 0/OFF states (in this order). Default colors are dark grey and light grey.

sepinfo An optional structure to form groups of genes in the plot. This is a list with the following element

> **class** A vector of names for the groups. These names will be printed in the region belonging to the group in the plot.
>
> **index** A list with the same length as *class*. Each element is a vector of gene indices belonging to the group.

outfile The name of the output .tex file

title A string printed as the title of the plot

Details:

For each attractor in *attractorinfo*, a LaTeX table environment is created. The output file does not contain a document header and requires the inclusion of the packages *tabularx* and *colortbl*. The tables have the genes in the rows and the states of the attractor in the columns. If not specified otherwise, cells of the table are light grey for 0/OFF values and dark grey for 1/ON values. If *sepinfo* is set, the genes are rearranged according to the indices in the group, horizontal separation lines are plotted between the groups, and the group names are printed.

Value:

This function has no return value.

Example:

```
data(examplenet)
attractor = getattractor(net,"./attractor")
plotattractor2tex(attractor$attractor,net,outfile="attractors.tex")
```

topajek function

Exports a network to the Pajek file format to visualize transition trajectories. For more information, see `http://pajek.imfm.si`.

Usage:

```
topajek(tt, storefile)
```

Arguments:

tt A transition table to visualize. This is usually the *$table* component of the return value of a call to `getattractor()`.

storefile The name of the output file for Pajek.

Example:

```
data(examplenet)
attractor = getattractor(net,"./attractor")
topajek(attractor$table,"topajek")
```

References

[1] R. Albert and H. G. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. *Journal of Theoretical Biology*, 223(1):1–18, 2003.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland, Oxford, 4th edition, 2002.

[3] M. Davidich and S. Bornholdt. The transition from differential equations to Boolean networks: a case study in simplifying a regulatory network model. *Journal of Theoretical Biology*, 255(3):269–277, 2008.

[4] H. de Jong. Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. *Journal of Computational Biology*, 9(1):67–103, 2002.

[5] V. Devloo, P. Hansen, and M. Labbé. Identification of All Steady States in Large Networks by Logical Analysis. *Bulletin of Mathematical Biology*, 65:1025–1051, 2003.

[6] E. R. Dougherty, S. Kim, and Y. Chen. Coefficient of determination in nonlinear signal processing. *Signal Processing*, 80(10):2219–2235, 2000.

[7] A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 2006.

[8] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[9] C. Gershenson, S. A. Kauffman, and I. Shmulevich. The Role of Redundancy in the Robustness of Random Boolean Networks. In *Artificial Life X, Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*. MIT Press, 2006.

[10] D. Gilbert, H. Fu, X. Gu, R. Orton, S. Robinson, V. Vyshemirsky, M. J. Kurth, C. S. Downes, and W. Dubitzky. Computational methodologies for modelling, analysis and simulation of signalling networks. *Briefings in Bioinformatics*, 7(4):339–353, 2006.

[11] T. E. Ideker, V. Thorsson, and R. M. Karp. Discovery of regulatory interactions through perturbation: inference and experimental design. In *Proceedings of the Pacific Symposium on Biocomputing 5*, pages 302–313, 2000.

[12] G. Karlebach and R. Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology*, 9:770–780, 2008.

[13] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[14] S. A. Kauffman. *The Origins of Order – Self-organization and Selection in Evolution.* Oxford University Press, New York, 1993.

[15] H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the boolean network model. *Machine Learning*, 52(1-2):147–167, 2003.

[16] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang. The yeast cell-cycle network is robustly designed. *PNAS*, 101(14):4781–4786, 2004.

[17] T. MacCarthy, A. Pomiankowski, and R. Seymour. Using large-scale perturbations in gene network reconstruction. *BMC Bioinformatics*, 5(11), 2005.

[18] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2008. `http://www.R-project.org`.

[19] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

[20] I. Shmulevich, I. Gluhovsky, R. F. Hashimoto, E. R. Dougherty, and W. Zhan. Steady-state analysis of genetic regulatory networks modelled by probabilistic Boolean networks. *Comparative and Functional Genomics*, 4(6):601–608, 2003.

[21] I. Wegener. *The Complexity of Boolean Functions.* B. G. Teubner, and John Wiley & Sons, 1987.

[22] Y. Xiao and E. R. Dougherty. The impact of function perturbations in Boolean networks. *Bioinformatics*, 23(10):1265–1273, 2007.

[23] S.-Q. Zhang, M. Hayashida, T. Akutsu, W.-K. Ching, and M. K. Ng. Algorithms for Finding Small Attractors in Boolean Networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05	*V. Arvind, J. Köbler, R. Schuler* On Helping and Interactive Proof Systems
94-06	*Christian Kalus, Peter Dadam* Incorporating record subtyping into a relational data model
94-07	*Markus Tresch, Marc H. Scholl* A Classification of Multi-Database Languages
94-08	*Friedrich von Henke, Harald Rueß* Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge
94-09	*F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker* Construction and Deduction Methods for the Formal Development of Software
94-10	*Axel Dold* Formalisierung schematischer Algorithmen
94-11	*Johannes Köbler, Osamu Watanabe* New Collapse Consequences of NP Having Small Circuits
94-12	*Rainer Schuler* On Average Polynomial Time
94-13	*Rainer Schuler, Osamu Watanabe* Towards Average-Case Complexity Analysis of NP Optimization Problems
94-14	*Wolfram Schulte, Ton Vullinghs* Linking Reactive Software to the X-Window System
94-15	*Alfred Lupper* Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen
94-16	*Robert Regn* Verteilte Unix-Betriebssysteme
94-17	*Helmuth Partsch* Again on Recognition and Parsing of Context-Free Grammars: Two Exercises in Transformational Programming
94-18	*Helmuth Partsch* Transformational Development of Data-Parallel Algorithms: an Example
95-01	*Oleg Verbitsky* On the Largest Common Subgraph Problem
95-02	*Uwe Schöning* Complexity of Presburger Arithmetic with Fixed Quantifier Dimension
95-03	*Harry Buhrman,Thomas Thierauf* The Complexity of Generating and Checking Proofs of Membership
95-04	*Rainer Schuler, Tomoyuki Yamakami* Structural Average Case Complexity
95-05	*Klaus Achatz, Wolfram Schulte* Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06	*Christoph Karg, Rainer Schuler* Structure in Average Case Complexity
95-07	*P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe* ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen
95-08	*Jürgen Kehrer, Peter Schulthess* Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik
95-09	*Hans-Jörg Burtschick, Wolfgang Lindner* On Sets Turing Reducible to P-Selective Sets
95-10	*Boris Hartmann* Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper
95-12	*Klaus Achatz, Wolfram Schulte* Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists
95-13	*Andrea Mößle, Heiko Vogler* Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes
95-14	*Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß* A Generic Specification for Verifying Peephole Optimizations
96-01	*Ercüment Canver, Jan-Tecker Gayen, Adam Moik* Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE
96-02	*Bernhard Nebel* Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class
96-03	*Ton Vullinghs, Wolfram Schulte, Thilo Schwinn* An Introduction to TkGofer
96-04	*Thomas Beuter, Peter Dadam* Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering
96-05	*Gerhard Schellhorn, Wolfgang Ahrendt* Verification of a Prolog Compiler - First Steps with KIV
96-06	*Manindra Agrawal, Thomas Thierauf* Satisfiability Problems
96-07	*Vikraman Arvind, Jacobo Torán* A nonadaptive NC Checker for Permutation Group Intersection
96-08	*David Cyrluk, Oliver Möller, Harald Rueß* An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction
96-09	*Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*

Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12	*Gerhard Schellhorn* Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers
98-13	*Gerhard Schellhorn, Wolfgang Reif* Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers
98-14	*Mohammad Ali Livani* SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN
98-15	*Mohammad Ali Livani, Jörg Kaiser* Predictable Atomic Multicast in the Controller Area Network (CAN)
99-01	*Susanne Boll, Wolfgang Klas, Utz Westermann* A Comparison of Multimedia Document Models Concerning Advanced Requirements
99-02	*Thomas Bauer, Peter Dadam* Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation
99-03	*Uwe Schöning* On the Complexity of Constraint Satisfaction
99-04	*Ercument Canver* Model-Checking zur Analyse von Message Sequence Charts über Statecharts
99-05	*Johannes Köbler, Wolfgang Lindner, Rainer Schuler* Derandomizing RP if Boolean Circuits are not Learnable
99-06	*Utz Westermann, Wolfgang Klas* Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets
99-07	*Peter Dadam, Manfred Reichert* Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik '99
99-08	*Vikraman Arvind, Johannes Köbler* Graph Isomorphism is Low for ZPP^{NP} and other Lowness results
99-09	*Thomas Bauer, Peter Dadam* Efficient Distributed Workflow Management Based on Variable Server Assignments
2000-02	*Thomas Bauer, Peter Dadam* Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT
2000-03	*Gregory Baratoff, Christian Toepfer, Heiko Neumann* Combined space-variant maps for optical flow based navigation
2000-04	*Wolfgang Gehring* Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05	*Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel* Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos
2000-06	*Wolfgang Reif, Gerhard Schellhorn, Andreas Thums* Fehlersuche in Formalen Spezifikationen
2000-07	*Gerhard Schellhorn, Wolfgang Reif (eds.)* FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification
2000-08	*Thomas Bauer, Manfred Reichert, Peter Dadam* Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen
2000-09	*Thomas Bauer, Peter Dadam* Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT
2000-10	*Thomas Bauer, Manfred Reichert, Peter Dadam* Adaptives und verteiltes Workflow-Management
2000-11	*Christian Heinlein* Workflow and Process Synchronization with Interaction Expressions and Graphs
2001-01	*Hubert Hug, Rainer Schuler* DNA-based parallel computation of simple arithmetic
2001-02	*Friedhelm Schwenker, Hans A. Kestler, Günther Palm* 3-D Visual Object Classification with Hierarchical Radial Basis Function Networks
2001-03	*Hans A. Kestler, Friedhelm Schwenker, Günther Palm* RBF network classification of ECGs as a potential marker for sudden cardiac death
2001-04	*Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm* Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion
2002-01	*Stefanie Rinderle, Manfred Reichert, Peter Dadam* Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata
2002-02	*Walter Guttmann* Deriving an Applicative Heapsort Algorithm
2002-03	*Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk* A Mechanically Verified Compiling Specification for a Realistic Compiler
2003-01	*Manfred Reichert, Stefanie Rinderle, Peter Dadam* A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks
2003-02	*Stefanie Rinderle, Manfred Reichert, Peter Dadam* Supporting Workflow Schema Evolution By Efficient Compliance Checks
2003-03	*Christian Heinlein* Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04	*Stefanie Rinderle, Manfred Reichert, Peter Dadam* On Dealing With Semantically Conflicting Business Process Changes.
2003-05	*Christian Heinlein* Dynamic Class Methods in Java
2003-06	*Christian Heinlein* Vertical, Horizontal, and Behavioural Extensibility of Software Systems
2003-07	*Christian Heinlein* Safely Extending Procedure Types to Allow Nested Procedures as Values (Corrected Version)
2003-08	*Changling Liu, Jörg Kaiser* Survey of Mobile Ad Hoc Network Routing Protocols)
2004-01	*Thom Frühwirth, Marc Meister (eds.)* First Workshop on Constraint Handling Rules
2004-02	*Christian Heinlein* Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures
2004-03	*Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)* Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence
2005-01	*Armin Wolf, Thom Frühwirth, Marc Meister (eds.)* 19th Workshop on (Constraint) Logic Programming
2005-02	*Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven* 2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm
2005-03	*Walter Guttmann, Markus Maucher* Constrained Ordering
2006-01	*Stefan Sarstedt* Model-Driven Development with ACTIVECHARTS, Tutorial
2006-02	*Alexander Raschke, Ramin Tavakoli Kolagari* Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen
2006-03	*Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari* Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg
2006-04	*Thorsten Liebig* Reasoning with OWL - System Support and Insights –
2008-01	*H.A. Kestler, J. Messner, A. Müller, R. Schuler* On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03	*Alena Hallerbach, Thomas Bauer, Manfred Reichert* Correct Configuration of Process Variants in Provop
2009-04	*Martin Bader* On Reversal and Transposition Medians
2009-05	*Barbara Weber, Andreas Lanz, Manfred Reichert* Time Patterns for Process-aware Information Systems: A Pattern-based Analysis
2009-06	*Stefanie Rinderle-Ma, Manfred Reichert* Adjustment Strategies for Non-Compliant Process Instances
2009-07	*H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid* Statistical Computing 2009 – Abstracts der 41. Arbeitstagung
2009-08	*Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam* Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen
2009-09	*Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch* Progressive Point-Light-Based Global Illumination
2009-10	*Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler* Boolean networks for modeling and analysis of gene regulation

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm