Evaluating the Reliability of Defect-Tolerant Architectures for Nanotechnology with Probabilistic Model Checking

Gethin Norman, David Parker, Marta Kwiatkowska
School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK

Sandeep K. Shukla
Bradley School of Electrical & Computer Engineering, Virginia Tech, Blacksburg, VA 24060, USA

Abstract

As we move from deep submicron technology to nanotechnology for device manufacture, the need for defect-tolerant architectures is gaining importance. This is because, at the nanoscale, devices will be prone to errors due to manufacturing defects, ageing, and transient faults. Micro-architects will be required to design their logic around defect tolerance through redundancy. However, measures of reliability must be quantified in order for such design methodologies to be acceptable. We propose a CAD framework based on probabilistic model checking which provides efficient evaluation of the reliability/redundancy trade-off for defect-tolerant architectures. This framework can model probabilistic assumptions about defects, easily compute reliability figures and help designers make the right decisions. We demonstrate the power of our framework by evaluating the reliability/redundancy trade-off of a canonical example, namely NAND multiplexing. We not only find errors in analytically computed bounds published recently, but we also show how to use our framework to evaluate various facets of design trade-off for reliability.

1. Introduction

With the continuing advances in the miniaturisation of devices, we are already at the deep submicron scale of device manufacture. However, nanotechnology is emerging as the technology of the not too distant future. In the nano era, device sizes will be in the range of several nanometres, leading to a high degree of failures, due to manufacturing defects, transient faults resulting from reduced noise tolerance at low voltage and current levels, and faults due to ageing because of molecular and other kinds of techniques for creating nano-devices. Although nano-scale manufacturing will allow us to pack more devices on a chip, we have to live with the possibilities of defects in the nano-substrate. As a result, 'defect-tolerant architecture' is being posed as a way to mitigate the challenge of the inherent unreliability at the nano-scale [6, 4, 9, 12]. Defect-tolerance is built into the architecture in the form of redundancy of devices and functional units. The resulting abundance of devices is acceptable due to their miniaturisation. However, it turns out that the redundancy level must be properly designed in order to obtain reliable computation from defect-prone devices.

Several theoretical models have been proposed in the past [4, 9] and results from information theory [10, 3] have been used to obtain some theoretical insight into these redundancy-reliability trade-offs. In contrast, our work is based on formal methods. With the help of PRISM [7, 11], a probabilistic model checking tool, we present a robust experimental platform that will allow us to gain insight into defect-tolerant designs and their reliability measures.

We present results which demonstrate the advantages of using probabilistic model checking, to analyse defect-tolerant architectures. We chose to investigate NAND multiplexing [13], but this approach can be applied to other defect-tolerant architectures such as *R*-fold Modular Redundancy [13] and Reconfiguration [6, 8]. One difference between this work and the standard approaches is that we evaluate the reliability of specific cases as opposed a general framework, and hence are not restricted by the analytical bounds of reliability, e.g., see [13, 10]. Our results also show that, using this approach, it is straightforward to investigate the effect on reliability of slight variations in the behaviour of the system's components, e.g. the change in reliability as the probability of gate failure varies. Also, through the construction of a formal specification, required in the probabilistic model checking approach, we found a flaw in the the analytical approach of [4].

The next section introduces the background material and Section 3 describes how we use PRISM to model NAND muliplexing. Section 4 reports on the results obtained for this case study and Section 5 concludes the paper.

2. Background

In this section, we briefly outline concepts from defect-tolerant architectures for nanotechnology, NAND multiplexing, probabilistic model checking and the PRISM tool.

2.1. Defect-tolerant computing

In 1952, von Neumann studied the problem of constructing reliable computation from unreliable devices (due to unreliable valve based computers at that time), introducing a redundancy technique called NAND multiplexing [13]. He showed that, if the failure probabilities of the gates are sufficiently small and failures are independent, then computations may be done with a high probability of correctness. Later, in [2], it was shown that a logarithmic redundancy is necessary for some Boolean function computations, and sufficient for all Boolean functions. Pippenger [10] showed that von Neumann's construction works only when the probability of failure per gate is strictly less than $1/2$, and that computation in the presence of noise (which can be seen as the presence of defect), requires more layers of redundancy. In [4, 9], NAND multiplexing was compared to other techniques for fault-tolerance and theoretical calculations showed that the redundancy level must be quite high to obtain acceptable levels of reliability.

Formally, a *defect-tolerant architecture* is one which uses techniques to mitigate the effects of defects in the devices that make up the architecture, and guarantees a given level of reliability.

2.2. Multiplexing

The basic technique of multiplexing is to replace a processing unit by a multiplexed unit with N copies of every input and output of the processing unit. In a multiplexing unit, there are N devices which in parallel process the copies of the inputs to give N outputs. If the inputs and devices are reliable, then each element of the output set will be identical and equal to that of the processing unit. However, when there are errors in the inputs and devices are faulty, the outputs will not be identical. Instead, after defining some critical level $\Delta \in (0, 0.5)$, the output of the multiplexing unit is considered stimulated (taking logical value `true`) if at least $(1-\Delta)\cdot N$ of the outputs are stimulated and non-stimulated (taking logical value `false`) if no more than $\Delta\cdot N$ outputs are stimulated. In cases where the number of stimulated outputs does not meet either criteria, i.e. the number of stimulated outputs is in the interval $(\Delta\cdot N, (1-\Delta)\cdot N)$, then the output is undecided, and hence a malfunction will occur.

The basic design of a multiplexing unit consists of two stages: the *executive stage* which performs the basic function of the processing unit to be replaced, and the *restorative stage* which reduces the degradation in the executive stage caused by errors in both the inputs and faulty devices.

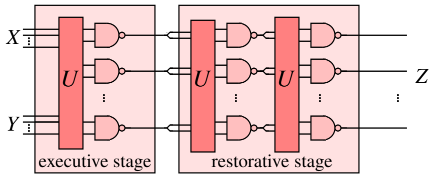

Figure 1. A NAND multiplexing unit

2.2.1. NAND multiplexing We now consider multiplexing when the processing unit is a single NAND gate. We therefore replace the inputs and output of the gate with N copies and in the executive stage duplicate the NAND gate N times, as in Figure 1. The unit U represents a *random permutation* of the input signals, that is, each signal of the first input is randomly paired with a signal from the second input to form an input pair for one of the copies of the gate. Also shown in Figure 1 is the restorative stage which takes the output of the executive stage as its inputs. Note that, applying this approach only once will invert the result, therefore two steps are required. To give a more effective restoration mechanism these stage can be iterated [13].

In [13], von Neumann concluded that, for extremely large N, the number of stimulated outputs of the executive stage is a stochastic variable, approximately normally distributed, and gave an upper bound of 0.0107 for the probability of gate failure that can be tolerated. In other words, according to von Neumann, if the failure probability is greater than this threshold, then the probability of the NAND multiplexing system failing will always be larger than a fixed, positive lower bound. Recently, it was shown that, if each NAND gate fails independently, the tolerable threshold probability of each gate is 0.08856 [3]. In [4] it is shown that, for small N, the number of outputs of the executive stage is theoretically a binomial distribution.

2.3. Probabilistic model checking and PRISM

Probabilistic model checking refers to a range of techniques for calculating the likelihood of the occurrence of certain events during the execution of systems, and can be useful to establish performance measures such as 'shutdown occurs with probability at most 0.1' and 'the video frame is delivered within 5ms with probability at least 0.9'.

We use PRISM [7, 11], a probabilistic model checker developed at the University of Birmingham. It supports analysis of: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). The models used here are DTMCs, which specify the probability of transitions between states, such that the probabilities of transitions from a given state sum up to 1. We also mention MDPs which extend DTMCs by allowing non-deterministic behaviour. Non-determinism enables the modelling of the asynchronous parallel composition of probabilistic systems and permits the under-specification of certain aspects of a system's behaviour.

In PRISM, these models are described in a high-level language based on guarded commands. A system is constructed as a number of modules which interact by means of standard process algebraic operations. A module contains a number of variables which represents its state. Its behaviour is given by a set of guarded commands of the form:

[] <guard> → <command>;

The guard is a predicate over the variables of the system and the command describes a transition which the module can make if the guard is true If a transition is probabilistic, then the command is specified as:

<prob> : <command> + ⋯ + <prob> : <command>

PRISM accepts specifications in the temporal logics PCTL [5] and CSL [1]. In the case of DTMCs, the model type used in this paper, specifications are written in PCTL, and for the analysis PRISM implements the algorithms of [5].

3. NAND multiplexing and PRISM

In this section we outline our approach of using probabilistic model checking to investigate the reliability of defect-tolerant architectures for nanotechnology. More precisely, we explain how PRISM can be used to model and analyse the performance of NAND multiplexing. We obtain very interesting results that not only uncover a bug in the analytical methods of [4], but also reveal interesting trends as the gate failure probability varies. This demonstrates that, we can not only quickly evaluate performance, but can also easily find counter-intuitive phenomena.

Note that the results presented here differ from theoretical lower bounds on the failure rates for correctness, since our results are with respect to fixed configurations. The advantage of our approach is that *exact* figures for the configuration are obtained. The disadvantage is that the results do not carry over to architectures with different configurations. However, one can simply construct a PRISM model for a different configuration and run experiments on this model.

3.1. Model construction

In this section we explain the PRISM model. We first note that it was during this phase that we noticed the error made by [4] in modelling the random permutation made by the unit U: in their analysis technique U in fact performs a random choice *with* replacement. More precisely, in the approach of [4], if in the previous stage there are k stimulated outputs, then after passing through the unit U the probability that any one of the resulting inputs is stimulated is k/N. As our analysis will demonstrate, these different approaches will lead to different conclusions about reliability. We should note however, that, as the bundle size increases, the differences between the results obtained by the approaches will converge and, in fact, if the bundle sizes were infinite then these approaches would be equivalent.

```
const N=20; // number of inputs in each bundle
const M=3; // number of restorative stages equals (M − 1)/2
prob p_err=0.01; // probability gate has von Neumann error
prob p_in=0.9; // probability an input is stimulated

module multiplex system

u : [1..M]; // current stage
c : [0..N] init N; // counter
s : [0..3]; // local state
n_x : [0..N]; // number of stimulated X inputs
n_y : [0..N]; // number of stimulated Y inputs
x : [0..1]; // value of first input
y : [0..1]; // value of second input
z : [0..N]; // number of stimulated outputs

// move to next NAND gate of current stage
[] s=0∧c>0→(s'=1);
// move onto next stage
[] s=0∧c=0∧u<M→
        (s'=1)∧(n_x'=z)∧(n_y'=z)∧(z'=0)∧(u'=u+1)∧(c'=N);
// initial choice of x and y: random choice
[] s=1∧u=1 → p_in : (x'=1)∧(s'=2)+(1−p_in) : (x'=0)∧(s'=2);
[] s=2∧u=1 → p_in : (y'=1)∧(s'=3)+(1−p_in) : (y'=0)∧(s'=3);
// choice x and y: random permutation
[] s=1∧u>1 → n_x/c : (x'=1)∧(s'=2)∧(n_x'=n_x−1)
         +(c−n_x)/c : (x'=0)∧(s'=2);
[] s=2∧u>1 → n_y/c : (y'=1)∧(s'=3)∧(n_y'=n_y−1)
         +(c−n_y)/c : (y'=0)∧(s'=3);
// NAND gate (with von Neumann error)
[] s=3 → p_err : (z'=z+(x∧y))∧(s'=0)∧(c'=c−1)
   +(1−p_err) : (z'=z+!(x∧y))∧(s'=0)∧(c'=c−1);

endmodule
```

Figure 2. PRISM description of the unit

Note that, unlike when U performs a random choice with replacement, if U performs a random permutation, the inputs of each gate in a stage are dependent on one other. Therefore, in this scenario, is not as straightforward to calculate the reliability of a NAND multiplexing unit by analytical techniques as, e.g. in [4]. However, as far as a probabilistic model checker is concerned, the only requirement is that the user correctly specifies the behaviour.

We now explain the main steps in the construction of our PRISM model. The first approach was to directly model the system as in Figure 1: for each stage construct a PRISM module for each gate and combine the modules through synchronous parallel composition. However, following this approach leads to the well know state space explosion problem, e.g. for a I/O bundle size of 20, modelling just the executive stage requires more than 10^{14} states.

The main observation, which allowed us to overcome this problem, was that the actual value of each input and output is not important: one need only store the total number of stimulated inputs and outputs. Furthermore, to allow us to compute these values, without having to store the output of each gate, we replace the N gates working in *parallel* with N gates working in *sequence*. This allows us to fold space into time, i.e. reuse the same gate over time rather than making redundancy over space. Furthermore, one can apply the same methodology to the stages, i.e. reuse the same module for each stage while keeping track of the outputs from the previous stage. Note that taking this approach does not influence performance since each gate works independently.

Following these observations, the PRISM model of the NAND multiplexing system, is given in Figure 2. To change the number of stages, bundle size, input distribution or gate failure probability requires only modification of the constants given in the description. Since PRISM can also represent non-determinism, one could set bounds on the probability of gate failure and then obtain (best and worst case) reliability characteristics for the system under these bounds.

4. Experimental results

In this section we study the performance of NAND multiplexing systems when the I/O bundles size is 20 and 40. In all the experiments, we assume that the inputs X and Y are identical (this is often true in circuits containing similar devices) and that initially 90% of the inputs are stimulated. We suppose the failure in each gate is a von Neumann fault, i.e. when a gate fails, the value of its output is inverted.

The properties we consider are the probability of there being k stimulated outputs (which in terms of the PRISM model in Figure 2, corresponds to the probability of reaching the state with z=k, u=M and c=N). By performing this analysis we in fact compute the output distribution of the system, and hence any measure of reliability can be calculated from these results. Note that PRISM can be used directly for computing the measures of reliability we consider.

Experiments were run on a PC running Linux, with a 1400 MHz processor and 512 MB of RAM. For a bundle size of 20 (model size: 78,311 states), it took PRISM 1.4 seconds to construct the model and 3.3 seconds to calculate the probability of there being 0 stimulated outputs. When the bundle size is 40 (model size: 1,004,821 states), model construction required 5.4 seconds and calculating the probability of there being 0 stimulated outputs took 28 seconds.

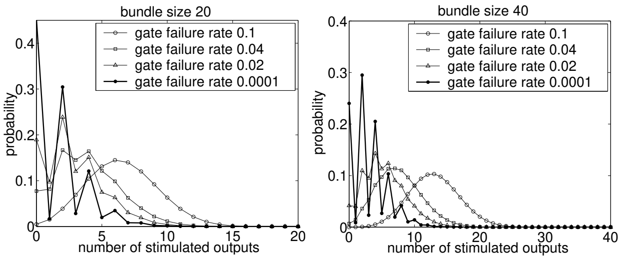

Figure 3. Output distributions

Our analysis of the reliability of the NAND multiplexing system concentrates on the effects of changing the failure probabilities of the NAND gates and the number of restorative stages. The results we present show:

- the shape of the output distribution as the probability of gate failure varies (Figure 3);
- an analysis of reliability, in terms of the probability that at most 10% of the outputs are incorrect, as the probability of gates failure varies (Figure 4);
- for different probabilities of gate failure, the resulting change in shape of the output distribution when additional restorative stages are added (Figure 5);
- how, in the case when the probability of gate failure is very small, the reliability can be improved by increasing the number of restorative stages (Figure 4);
- by comparing the probability that at most 10% of the outputs are incorrect and the expected percentage of incorrect outputs for different numbers of stages, the maximum probability of gate failure allowed for the system to function reliably (Figures 6 and 7).

Where appropriate, we also compare the results with those obtained when, as in [4], the unit U performs a random choice with replacement (referenced 'UR').

4.1. Basic system reliability

Initially, we consider the basic version of the NAND multiplexing system shown in Figure 1. We first investigate the effect of changing the failure probabilities of the gates has on the reliability of the system. Figure 3 presents the output distribution of the system in the cases when the bundle size is 20 and 40 and the probability of gate failure equals 0.1, 0.04, 0.02 and 0.0001. Note that the system is supposed to model a correctly functioning NAND gate and we assume that the inputs are correct when stimulated. Hence, the more outputs that are non-stimulated, the greater the reliability.

Using the distributions, in Figure 4 (1 restorative stage), we have plotted the probability that less than 10% of the outputs are incorrect. We also plot the results when U performs a random choice with replacement ('UR'). As expected, both the distributions in Figure 3 and the results in

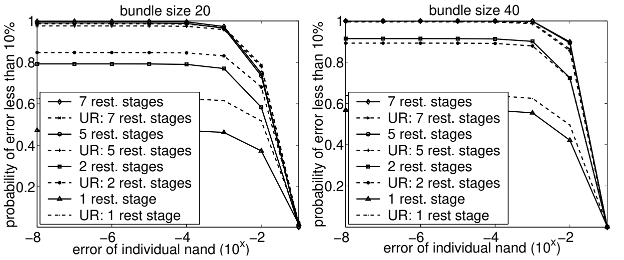

Figure 4. Reliability versus gate failure

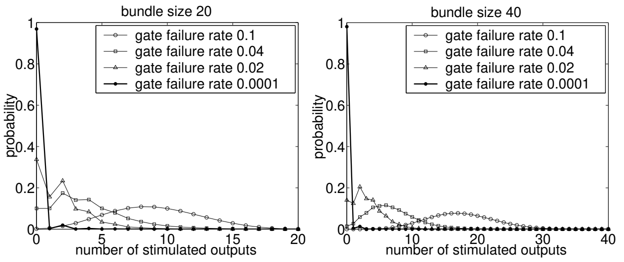

Figure 5. Output dist. (4 restorative stages)

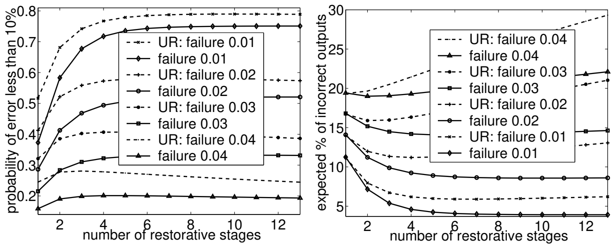

Figure 6. Reliability versus stages ($N = 20$)

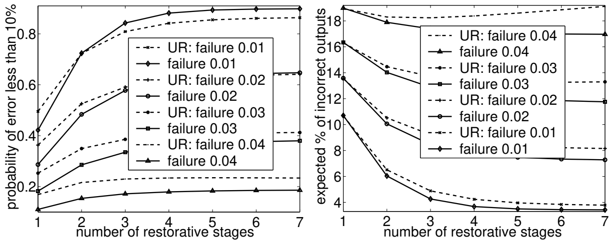

Figure 7. Reliability versus stages ($N = 40$)

Figure 4 show that, as the gate failure probability decreases, reliability increases. Also, Figure 4 demonstrates that increasing the bundle size decreases the chance of error.

Considering Figure 4 in the case when U performs a random choice with replacement (denoted 'UR'), we see that this leads to an over approximation of the reliability of the multiplexing system: the chance of correct outputs is higher than when U is modelled correctly. Note that, as our results will demonstrate, modelling U in this way does not always lead to upper bounds on the reliability.

4.2. Adding restorative stages

Next, we investigate the change in reliability of a NAND multiplexing system as more restorative stages are added to the system. In Figure 5, we present the output distribution of the system with 4 restorative stages. Comparing these distributions with those in Figure 3, we see that, when the gate failure probability is sufficiently small (0.0001), adding restorative stages results in a far more reliable system (the probability of any stimulated outputs is very small). On the other hand, if the probability of gate failure is large (0.1), then adding stages does not increase reliability and, in fact, can actually decrease the reliability of the system.

4.2.1. Small probabilities of gate failure To emphasise the first observation in the previous paragraph, Figure 4 presents, for small gate failure probabilities, the probability that at most 10% of the outputs of the overall system are incorrect against the gate failure probability. The results again demonstrate that increasing the number of restorative stages can enhance the reliability. However, they also show that the rate of increase in reliability decreases as more stages are added, and that there is a limit to the reliability which can be gained by adding stages: compare the plots presented in Figure 4 when the number of restorative stages is 5 and 7. We should also mention that this result corresponds to the observation made in [4] that, as the number of stages increases, the output distribution of the system will eventually become stable and independent of the number of stages.

In Figure 4, we have also included the statistics obtained when the unit U performs a random choice with replacement instead of a random permutation. In this case, unlike in our earlier results, the approach no longer gives either an upper or lower bound on the reliability of the system. As mentioned earlier, for the larger bundle size (40), the difference between the results with the two approaches decreases.

4.2.2. Large probabilities of gate failure We now consider the case when the probability of gate failure of the NAND gates becomes too large for the system to function reliably. In Figures 6 and 7, we have plotted both the probability that system error is less than 10% and the expected percentage of incorrect inputs against the number of restorative stages when the failure rate of the NAND gates is between 0.04 and 0.01.

As can be seen from Figure 6, when the bundle size is 20, and the probability of gate failure equals 0.04, increasing the number of restorative stages cannot make the computation reliable and, in fact, if one keeps increasing the stages, the reliability actually starts to decrease. This anomalous behaviour can be understood as follows: when the failure rate is 0.04, the restorative stages are sufficiently affected by the probability of gate failures to be unable to reduce the degradation, and hence increasing number of stages makes the system more amenable to error. From the results we therefore conclude that, for a bundle size of 20, if the gate failure probability is 0.04, then the system cannot be made reliable. While, if the failure probability is 0.01, then, for certain criteria, the system can be made reliable once a sufficient number of restorative stages have been added.

When the bundle size is 40, Figure 7 show that, if the gate failure probability is 0.04, then the system can be classified as unreliable no matter how many restorative stages are added. However, when the failure probability is 0.01 (and for certain criteria 0.02), the system can be considered reliable once a sufficient number of stages are added.

In Figures 6 and 7, statistics when the unit U performs a random choice with replacement are again included. The results show that, this approach can lead to different results, e.g. in Figure 6 when the gate failure probability equals 0.03, if U performs a random choice with replacement, then adding stages increases reliability, whereas if U (correctly) performs a permutation, reliability decreases. Considering the results in Figure 7, when the probability of gate failure is 0.01 or 0.02 and the number of stages is small, supposing U performs a random choice with replacement leads one to assume that the system is more reliable than it is, however, as the number of stages increases, the converse holds.

5. Conclusion and future work

In this work, we have demonstrated how probabilistic model checking can be used for the evaluation of the redundancy/reliability trade-off in defect-tolerant architectures for nanotechnology. In particular, we have shown how, for a given probability of gate failure, probabilistic model checking can find the minimum level of redundancy (I/O bundle size and number of stages) to enable reliable computation.

We have reported on the results obtained for NAND multiplexing and investigated the performance as the error rate of the individual NAND gates and the number of stages varies. The first step in this approach involved constructing a system model in PRISM's input description language, and we described an approach which reduced the effect of the well known state space explosion problem, and hence allowed for the analysis of larger configurations. In the analysis using PRISM, we were able to compute the exact output distribution of the system, and hence construct a complete picture of the reliability of the system under study.

We chose to analyse NAND multiplexing as it is a canonical example which is standard in the literature, and enabled us to compare the techniques and results with others. However, as explained, this approach can equally be applied to alternative fault-tolerant architectures for nanotechnology.

In conclusion, this paper shows how probabilistic model checking offers a complementary approach to the theoretical results present in the literature for analysing defect-tolerant architectures for nanotechnology. More precisely, our analysis technique allows us to obtain sharp bounds and study probabilistic anomalies for a fixed architecture, as opposed to establishing bounds in the generic case.

Acknowledgements

This work was supported by NSF grants CCR-0237947, CCR-0340740 EPSRC grants GR/N22960 and GR/S11107, FORWARD and SRC.

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Proc. CAV'96*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[2] R. Dobrushin and E. Ortyukov. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Problems of Information Transmission*, 13(3):203–218, 1977.

[3] W. Evans and N. Pippenger. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 44(3):1299–1305, 1998.

[4] J. Han and P. Jonker. A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, 1:201–208, 2002.

[5] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[6] J. Heath, G. S. P. Kuekes, and R. Williams. A defect tolerant computer architecture: Opportunities for nanotechnology. *Science*, 80:1716–1721, 1998.

[7] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. TOOLS'02*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

[8] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Towards robust integrated circuits: The embryonics approach. *Proc. IEEE*, 88(4):516–541, 2000.

[9] K. Nikolic, A. Sadek, and M. Forshaw. Architectures for reliable computing with unreliable nanodevices. In *Proc. IEEE-NANO'01*, pages 254–259. IEEE, 2001.

[10] N. Pippenger. Reliable computation by formulas in the presence of noise. *IEEE Transactions on Information Theory*, 34(2):194–197, 1988.

[11] PRISM web page: www.cs.bham.ac.uk/~dxp/prism.

[12] M. Ratner and D. Ratner. *Nanotechnology: Gentle Introduction to the Next Big Idea*. Prentice Hall, 2003.

[13] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.