This is the author's version of the work. The definitive work was published in *Proceedings of Design Automation Conference (DAC 2010), 2010.* The work is supported in part by the German Science Foundation (DFG), SFB 694.

Towards Scalable System-Level Reliability Analysis*

Michael Glaß, Martin Lukasiewycz, Christian Haubelt, and Jürgen Teich
University of Erlangen-Nuremberg, Germany
{glass,martin.lukasiewycz,haubelt,teich}@cs.fau.de

ABSTRACT

State-of-the-art automatic reliability analyses as used in system-level design approaches mainly rely on Binary Decision Diagrams (BDDs) and, thus, face two serious problems: (1) The BDDs exhaust available memory during their construction and/or (2) the final size of the BDDs is, sometimes up to several orders of magnitude, larger than the available memory. The contribution of this paper is twofold: (1) A partitioning-based early quantification technique is presented that aims to keep the size of the BDDs during construction at minimum. (2) A SAT-assisted simulation approach aims to deliver approximated results when exact analysis techniques fail because the final BDDs exhaust available memory. The ability of both methods to accurately analyze larger and more complex systems than known approaches is demonstrated for various test cases.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

General Terms

Reliability, design

Keywords

Reliability analysis, SAT-assisted simulation, early quantification

1. INTRODUCTION

Although shrinking CMOS technology sizes allow to integrate more complex systems on a single chip and, thus, have advantages for important design objectives like monetary costs and volume consumption, the resulting increase in process variation is a major issue regarding the reliability of the designed components. The requirement to design reliable systems from these unreliable components has made *reliability* to become one of the main objectives of modern automatic embedded system-level design approaches.

Formal state-of-the-art reliability analysis techniques like [5, 8] are based on the abstraction of the system-level design to a Boolean function given as a *Binary Decision Diagram* (BDD). The exponential worst-case complexity of BDDs leads to two serious drawbacks for the scalability of the analysis approaches: (1) The BDDs exhaust available memory during their construction and/or (2) the size of the final BDDs is often up to several orders of magnitude larger than the available memory. These drawbacks are critical for the so-called *up-scaling*, i.e., combining several subsystems and components to form the complex overall system that results in an explosion of the required memory. To make a contribution towards scalable system-level reliability analysis techniques, the work at hand proposes two approaches to cope with the abovementioned problem of oversized BDDs.

Contributions. Several known automatic reliability analyses rely on the use of temporary variables during the construction of the BDDs. This arises for example from an analysis that only considers hardware component defects, but needs to consider the binding of processes to determine the *state of the system*, i.e., whether the system works properly or not. The variables used for the process binding are excluded using *exists quantification*, such that the final BDD reflects the reliability based on component defects. Thus, in contrast to the final BDD, the size of the temporary BDDs at construction time is critical. To cope with this problem, an *early quantification* approach based on a heuristic that is tailored for system-level reliability analysis is proposed that aims to quantify variables as early as possible to keep the size of the BDDs during construction small.

For highly complex problems, the proposed early quantification method might fail due to oversized BDDs as well. As a remedy, the work at hand proposes a novel approach where a simulation is assisted by a state-of-the-art SAT-solver to approximate the reliability. This efficient simulation allows to carry out a high number of simulation runs and, thus, allows to trade the memory consumption of the BDD-based approaches for runtime. The simulative approach is capable of achieving a very accurate reliability analysis with a reasonable overhead regarding runtime even for large and very complex systems where known exact methods fail.

The proposed methodologies are compared to state-of-the-art reliability analysis approaches on several test cases to give evidence of their scalability.

Outline. The rest of the paper is outlined as follows: Section 2 discusses related work. While the problem targeted in this paper is outlined in Sec. 3, the the proposed reliability analysis approaches are introduced in Sec. 4 and Sec. 5 . Section 6 presents experimental results before the paper is concluded in Sec. 7.

2. RELATED WORK

The importance of reliability as an objective in automatic embedded system design is supported by a large number of approaches that have been published in recent years. These approaches aim to increase the reliability of the system by checkpointing and dynamic voltage scaling [21], improved scheduling techniques, cf. [10, 22], reliability-aware selection of components [11], or introducing redundancy at component-level [13, 17, 20], or process-level [5, 6, 8]. However, one can observe that nearly all reliability-increasing techniques result in an increase of cost in one or even several objectives like runtime, monetary costs, area consumption, or power consumption. Thus, finding optimal system implementations with respect to multiple and often conflicting objectives requires an accurate analysis technique for each objective. Most of the approaches discussed so far either perform

*Supported in part by the German Science Foundation (DFG), SFB 694.

reliability analysis using simplified failure models like a constant failure probability [13], rely on so-called *series-parallel-structures* for the analysis that cannot model important embedded system design aspects like resource reuse [3], or target reliability analysis at lower levels of abstraction using extensive simulation [22]. State-of-the-art system-level reliability analyses are presented in [8] and [5] that can analyze arbitrary embedded systems at system-level using Boolean functions encoded in BDDs. The scalability of the methodology proposed in [5] is improved by an application-specific early quantification approach presented in [6]. In contrast, the work at hand proposes a generalized early quantification heuristic that even outperforms the approach from [6]. In [9], the approach from [8] is extended by using a specific variant of BDDs which perform better for the considered test cases but are still suffering from the drawbacks of common BDDs.

Early quantification is a technique that aims to cope with the size explosion of BDDs during their construction known from several domains, most prominently from formal verification, cf., e.g., [15]. A smorgasbord of approaches for early quantification is available like [12, 19] with [7] being the most closely related approach. In comparison, the approach proposed in the work at hand uses a divide-and-conquer heuristic based on finding cuts in a graph-based representation of the dependencies between relations. Moreover, the relations are not explicitly given, but are a result of the clustering of conjunctions of terms. This heuristic is especially tailored for Boolean functions that have a *Conjunctive Normal Form* (CNF) related structure as typical for formal system-level reliability analysis.

Monte-Carlo simulation has been widely studied as an appropriate method to perform reliability analysis for large systems with complex behavior, cf. [1]. To the best of the authors knowledge, this is the first SAT-assisted simulation approach that implements an efficient system-level reliability analysis.

3. PROBLEM DEFINITION

Reliability analysis approaches commonly require to determine the so-called *reliability function* $\mathcal{R} : \mathbb{R}^+ \to \mathbb{R}_{[0,1]}$ of the overall system that returns the probability of the life time τ_{LT} of the system being greater than a certain time τ:

$$\mathcal{R}(\tau) = \mathcal{P}[\tau_{LT} > \tau] \tag{1}$$

It holds that $\mathcal{R}(0) = 1$ and $\mathcal{R}(\infty) = 0$. The reliability function allows to compute all important reliability-related measures like *Mean-Time-To-Failure* (MTTF) or the *Mission-Time* (MT). To determine the reliability function, knowledge about the state of the system in case of failures and defects is needed. Thus, most formal reliability analysis techniques rely on the so-called *structure function* φ. This Boolean function $\varphi : \{0,1\}^{|X|} \to \{0,1\}$ takes a vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_{|X|})$ encoding the states of all system components X, i.e., $x_i = 1$ if a component i works properly and $x_i = 0$ if it failed, and returns the state of the system as 1 if the system works properly and 0 if the system failed, respectively.

The main challenge for reliability analysis techniques is the representation of φ by appropriate data structures that are mostly based on *Binary Decision Diagrams* (BDDs) [2]. An example on how to derive the reliability function from a BDD-encoded structure function can be found in [6] and is outlined as follows: Using a specific Shannon-decomposition as proposed in [16], the probability $\mathcal{P}$ of a proper working system at time τ is determined by traversing the BDD representing φ:

$$\mathcal{P}(\tau, \varphi) = \mathcal{R}_x(\tau) \cdot \mathcal{P}(\tau, \varphi_{|x=1}) + (1 - \mathcal{R}_x(\tau)) \cdot \mathcal{P}(\tau, \varphi_{|x=0}) \tag{2}$$

This function determines the probability of a structure function φ to evaluate to 1 at a given time τ, depending on the reliability function $\mathcal{R}_x(\tau)$ of each component x with $\mathcal{R}_x : \mathbb{R}^+ \to \mathbb{R}_{[0,1]}$. The desired reliability function $\mathcal{R}$ of the overall system is then given by:

$$\mathcal{R}(\tau) = \mathcal{P}(\tau, \varphi) \tag{3}$$

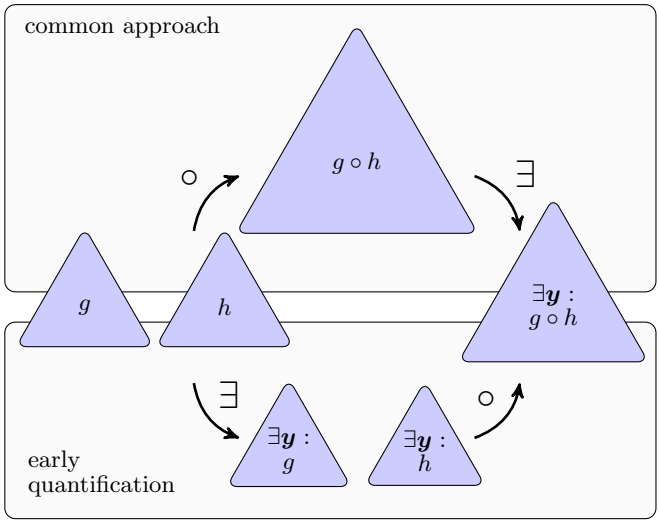


Figure 1: Early quantification aims to quantify temporary variables as early as possible to decrease the maximum size of the data structure during the construction process, cf. Eq. (5). The size of a triangle shall correspond to the size of the resulting data structure and $\circ = \{\wedge, \vee\}$.

4. EARLY QUANTIFICATION

In many automatic reliability analysis approaches, temporary variables Y represented by the binary vector $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i, \ldots, \boldsymbol{y}_{|Y|})$ are required in the structure function construction process to determine the correct state of the system, but are removed as soon as the overall structure is generated to derive φ:

$$\varphi(\boldsymbol{x}) = \exists \boldsymbol{y} : \psi(\boldsymbol{x}, \boldsymbol{y}) \tag{4}$$

Imagine the case where the state of the system depends on both, the availability of hardware components and on the ability of the system to correctly establish communication among processes. Thus, variables for the communicating processes Y are needed to construct $\psi(\boldsymbol{x}, \boldsymbol{y})$, with $\psi : \{0,1\}^{|X|+|Y|} \to \{0,1\}$. However, if the assumed failure model is the defect of components, the process variables in Y need to be *existentially quantified*[1] to derive the desired structure function $\varphi(\boldsymbol{x})$. The existential quantification ensures that $\varphi(\boldsymbol{x})$ evaluates to 1 only if there exists a feasible process binding that establishes a correct communication for a given state $\boldsymbol{x}$ of the hardware components. This leads to BDDs exhausting available memory during the construction process, although the BDD without the temporary variables might fit the memory.

This size explosion during the construction of the data structures is known from formal verification as well and can be faced using so-called *early quantification* techniques. These techniques aim to partition the problem into subproblems such that temporary variables can be quantified as soon as the construction of the BDD for each subproblem is completed. The resulting BDDs of the subproblems are combined afterwards. Given the function $s_{\max} : \{0,1\}^{\{0,1\}^n} \to \mathbb{N}$ that returns the maximum memory consumption of a

[1] $\exists y : \psi(\boldsymbol{x}, \boldsymbol{y})_{|y=1} \vee \psi(\boldsymbol{x}, \boldsymbol{y})_{|y=0}$

Boolean function[2] with n variables during the construction of the BDD, it can be observed that

$$s_{\max}(\exists \boldsymbol{y} : g(\boldsymbol{x},\boldsymbol{y}) \circ h(\boldsymbol{x},\boldsymbol{y})) \geq s_{\max}(\exists \boldsymbol{y} : g(\boldsymbol{x},\boldsymbol{y}) \circ \exists \boldsymbol{y} : h(\boldsymbol{x},\boldsymbol{y})). \quad (5)$$

with $\circ = \{\wedge, \vee\}$. That means, the earlier variables can be quantified during the construction of the BDD, the smaller is the maximum size of the BDD during construction. A schematic representation of early quantification is depicted in Fig. 1. When Boolean functions are combined using disjunction (logical *or*, $\vee$), early quantification becomes trivial since it holds:

$$\exists \boldsymbol{y} : (g(\boldsymbol{x},\boldsymbol{y}) \vee h(\boldsymbol{x},\boldsymbol{y})) \Leftrightarrow \exists \boldsymbol{y} : g(\boldsymbol{x},\boldsymbol{y}) \vee \exists \boldsymbol{y} : h(\boldsymbol{x},\boldsymbol{y}) \quad (6)$$

On the other hand, Boolean functions combined using conjunction (logical *and*, $\wedge$) are challenging with respect to early quantification. It holds:

$$\exists \boldsymbol{y} : (g(\boldsymbol{x},\boldsymbol{y}) \wedge h(\boldsymbol{x},\boldsymbol{y})) \Leftrightarrow \exists \boldsymbol{y} : g(\boldsymbol{x},\boldsymbol{y}) \wedge \exists \boldsymbol{y} : h(\boldsymbol{x},\boldsymbol{y}), \text{ iff } \forall y \in Y : \neg \mathrm{c}(g(\boldsymbol{x},\boldsymbol{y}), y) \vee \neg \mathrm{c}(h(\boldsymbol{x},\boldsymbol{y}), y) \quad (7)$$

with $\mathrm{c} : \{0,1\}^{\{0,1\}^{|X|+|Y|}} \times Y \rightarrow \{0,1\}$ that evaluates to 1 if a variable is *contained* in a function, i.e., if the function is not invariant to a variable, and 0 if the variable is not contained in the function, respectively. Thus, the condition states that an early quantification is only allowed if the variables to quantify are contained in at most the function $g(\boldsymbol{x},\boldsymbol{y})$ or $h(\boldsymbol{x},\boldsymbol{y})$. In most cases, the functions will share variables and, thus, Eq. (7) is not applicable. To overcome this problem, the work at hand proposes a transformation of Eq. (7) to allow early quantification based on a special partitioning of the variables Y. In this partitioning, three sets are determined such that two sets Y_g and Y_h consist of variables that are contained in one function only, while the third set Y_z consists of the variables that are shared by both functions. With corresponding subvectors $\boldsymbol{y^g}, \boldsymbol{y^h}, \boldsymbol{y^z}$ of $\boldsymbol{y}$ being defined, it holds:

$$\exists \boldsymbol{y} : (g(\boldsymbol{x},\boldsymbol{y}) \wedge h(\boldsymbol{x},\boldsymbol{y})) \Leftrightarrow \exists \boldsymbol{y^z} : (\exists \boldsymbol{y^g} : g(\boldsymbol{x},\boldsymbol{y}) \wedge \exists \boldsymbol{y^h} : h(\boldsymbol{x},\boldsymbol{y})) \quad (8)$$

$$\text{with} \quad (Y_z \cup Y_g \cup Y_h) = Y \wedge \forall y \in Y_g : \neg \mathrm{c}(h(\boldsymbol{x},\boldsymbol{y}), y) \wedge \forall y \in Y_h : \neg \mathrm{c}(g(\boldsymbol{x},\boldsymbol{y}), y) \wedge \forall y \in Y_z : \mathrm{c}(g(\boldsymbol{x},\boldsymbol{y}), y) \wedge \mathrm{c}(h(\boldsymbol{x},\boldsymbol{y}), y)$$

Following this early quantification scheme allows to early quantify the two functions g and h with respect to variables that are contained in only one function, combine the resulting data structures using conjunction, and finally quantify the variables contained in both functions. Thus, this approach is capable of making use of early quantification where former approaches like [6], where early quantification is only enabled if Eq. (7) is applicable, fail.

In several reliability analysis techniques, partitioning is used to speed-up the analysis process by preventing outsized BDDs. Commonly, these approaches rely on domain-specific knowledge about the structure of the system for the partitioning, cf. [6]. Given Eq. (8), the work at hand proposes a partitioning that partitions temporary variables based on a given Boolean function solely and, thus, does not need to take into account the given application and/or architecture.

For the partitioning and without loss of generality, a Boolean function $\psi(\boldsymbol{x},\boldsymbol{y})$ of the form $\psi(\boldsymbol{x},\boldsymbol{y}) = \bigwedge_{t \in T} t(\boldsymbol{x},\boldsymbol{y})$ is assumed since disjunctions can be trivially early quantified following Eq. (6). Each term $t \in T$ is a Boolean function

[2]the set of all Boolean functions with n variables is written as $\{0,1\}^{\{0,1\}^n} = \{\varphi | \varphi : \{0,1\}^n \rightarrow \{0,1\}\}$

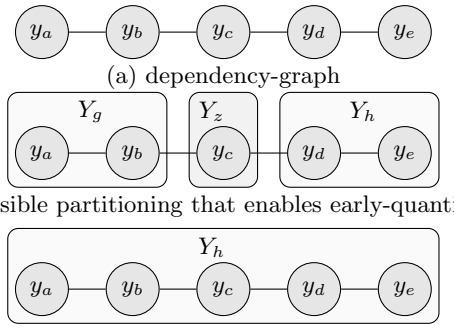


Figure 2: (a) The dependency graph for the Boolean function $\psi(\boldsymbol{x},\boldsymbol{y}) = (y_b \vee y_c) \wedge (y_c \vee y_d) \wedge (y_b \vee y_a) \wedge (y_d \wedge y_e)$ as well as two feasible partitions (b) $Y_g = \{y_b, y_a\}, Y_h = \{y_d, y_e\}, Y_z = \{y_c\}$ and (c) $Y_g = \emptyset, Y_h = \{y_a, y_b, y_c, y_d, y_e\}, Y_z = \emptyset$.

$t : \{0,1\}^{|X|+|Y|} \rightarrow \{0,1\}$. Given ψ in the above form, the structure function is defined as follows:

$$\varphi(\boldsymbol{x}) = \exists \boldsymbol{y} : \bigwedge_{t \in T} t(\boldsymbol{x},\boldsymbol{y}) \quad (9)$$

It is assumed that each term contains at least one variable $y \in Y$. Otherwise, these terms trivially fulfill Eq. (7) and, thus, can be analyzed independently.

For the partitioning, a *dependency graph* $D = (Y, E_y)$ is defined. Each node $y \in Y$ represents a temporary variable that shall be quantified. An edge $E_y = \{(y,\widetilde{y}) | y,\widetilde{y} \in Y \wedge \exists t \in T : c(t(\boldsymbol{x},\boldsymbol{y}), y) \wedge c(t(\boldsymbol{x},\boldsymbol{y}), \widetilde{y})\}$ between nodes is drawn if two variables y and $\widetilde{y}$ are contained in the same term of the given Boolean function.

Consider the following example of a Boolean function $\psi(\boldsymbol{x},\boldsymbol{y}) = (y_b \vee y_c) \wedge (y_c \vee y_d) \wedge (y_b \vee y_a) \wedge (y_d \wedge y_e)$. The dependency graph for ψ can be found in Fig.2(a). Following Eq. (8), several feasible partitions Y_g, Y_h, Y_z are available like $Y_g = \{y_a, y_b\}, Y_h = \{y_d, y_e\}, Y_z = \{y_c\}$, cf. Fig. 2(b), or $Y_g = \emptyset, Y_h = \{x_a, x_b, x_c, x_d, x_e\}, Y_z = \emptyset$, cf. Fig. 2(c). While the first partition can be considered good since it allows to early quantify 4 variables with only one variable being left for quantification after combining the functions, the latter partition can be considered bad since no variable can be quantified early. A good partitioning fulfills the following minimum-requirement:

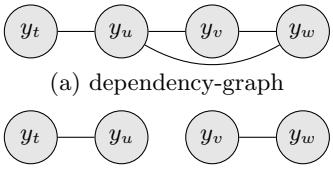
$$\min : \max(|Y_g| + |Y_z|, |Y_h| + |Y_z|) \quad (10)$$

In other words, a good partitioning aims to determine two large partitions Y_g and Y_h while trying to keep variables Y_z that are included in both partitions at minimum, such that many variables can be quantified early and only few variables are left for quantification after the functions are combined. This requirement allows a maximum benefit from the early quantification approach given in Eq. (8).

In the following, an algorithm is presented that makes use of the proposed early quantification scheme to construct the structure function φ.

4.1 Preprocessing

Given a Boolean function of the form given in Eq. (9), several well-known preprocessing techniques are applied. Important in this context is the rule of absorption that allows to eliminate terms. The effect can be visualized by the corresponding dependency graph before and after preprocessing: Due to absorption, edges $(y,\widetilde{y}) \in E_y$ can be removed if terms can be excluded such that two variables y and $\widetilde{y}$ are not contained in the same term anymore. The resulting graph has



(a) dependency-graph

(b) dependency-graph after preprocessing

Figure 3: The dependency graph for the Boolean function $\psi(\boldsymbol{x},\boldsymbol{y}) = (y_t \vee y_u) \wedge (y_v \vee y_w) \wedge (y_v \vee y_u \vee y_w)$ (a) before and (b) after the preprocessing.

Algorithm 1 dac(Y') - Divide-and-Conquer.

Require: $Y' \subseteq Y$
1: **if** $|Y'| < \epsilon$ **then return** $\exists \boldsymbol{y} : \bigwedge_{t \in \rho(Y')} t(\boldsymbol{x},\boldsymbol{y})$
2: **else**
3: $(Y_g, Y_h) :=$ divide(Y')
4: $g(\boldsymbol{x},\boldsymbol{y}) :=$ dac(Y_g)
5: $h(\boldsymbol{x},\boldsymbol{y}) :=$ dac(Y_h)
6: $Y_z := Y' \setminus (Y_g \cup Y_h)$
7: $T' := \rho(Y_z) \setminus (\rho(Y_g) \cup \rho(Y_h))$
8: $z(\boldsymbol{x},\boldsymbol{y}) := g(\boldsymbol{x},\boldsymbol{y}) \wedge h(\boldsymbol{x},\boldsymbol{y}) \wedge \bigwedge_{t \in T'} t(\boldsymbol{x},\boldsymbol{y})$
9: **return** $\exists \boldsymbol{y} : z(\boldsymbol{x},\boldsymbol{y})$
10: **end if**

a decreased problem complexity and, in some cases, decays to a set of independent subgraphs, cf. Fig. 3. These subgraphs can, following Eq. (7), be analyzed independently. In particular, the resulting subgraphs in the dependency graph correspond to the partitioning approach presented in [6]. However, in [6], domain-specific knowledge is used to find these independent subgraphs, based on a special analysis of the system during the construction of the Boolean function from the structure of the system. In contrast, the approach proposed in the work at hand is applicable to any given structure function.

4.2 Divide-and-Conquer Algorithm

In the following, an algorithm is used that recursively uses Eq. (8) to implement an efficient early quantification approach in a divide-and-conquer fashion: Given the set of components $X = \{x_1, \ldots, x_{|X|}\}$ of a Boolean function $\varphi(\boldsymbol{x})$, the set of temporary variables Y that are quantified, and the function $\rho : 2^{Y'} \rightarrow 2^T$ with $\rho(Y') = \{t | t \in T \wedge y \in Y' \wedge c(t,y)\}$ that allows to obtain all terms that contain at least one variable $y \in Y' \subseteq Y$, the approach can be formulated as follows:

$$\varphi(\boldsymbol{x}) = \text{dac}(Y) \tag{11}$$

The recursion is performed by the function *dac* that is outlined in Alg. 1. The algorithm requires the temporary variables that need to be considered for the current partition, cf. line 0. If the number of temporary variables in the partition is less than a given ϵ, the recursion ends and the partition is transformed into a BDD. If the partition contains too many temporary variables, the recursion works as follows: After the two partitions Y_g and Y_h are determined using the *divide* function, cf. line 3, the BDDs for the partitions Y_g and Y_h are constructed and early quantified following the recursion scheme, cf. lines 4 and 5. After Y_z is determined in line 6, the partitions are combined using conjunction in line 8 and a quantification with respect to the variables Y_z in line 9 completes the recursion.

4.3 Divide

As outlined in Eq. (10), determining a good partition is crucial for the effectiveness of the proposed early quantification approach. This partitioning is performed by the *divide* function. This function aims to find a set of candidate partitions and determines the best partitioning with respect to Eq. (10). Therefore, the *divide* algorithm performs an ordering of the nodes Y to derive an ordered set $P = (y_1 < \ldots < y_i < \ldots < y_{|Y|})$, cf. Alg. 2, first: The ordering algorithm starts with an empty ordered set P and generates the required dependency graph, cf. lines 0 and 1. It iteratively adds new nodes to the set with the next node to add being the one with the maximum *weight*, cf. lines 3 and 4. The weight w of a node is defined as follows:

$$w(y,P) = \begin{cases} \frac{w_i(y)}{w_o(y)}, & \text{if } w_o(y) > 0 \\ w_i(y), & \text{else.} \end{cases} \tag{12}$$

$$\text{with} \quad w_i(y) = |\{\widetilde{y} \,|\widetilde{y} \in Y \wedge (y,\widetilde{y}) \in E_y \wedge \widetilde{y} \in P\}|$$
$$w_o(y) = |\{\widetilde{y} \,|\widetilde{y} \in Y \wedge (y,\widetilde{y}) \in E_y \wedge \widetilde{y} \notin P\}|$$

Given the ordered set P, the *divide* function determines the best cut with respect to Eq. (10) by a linear search in the ordered set that results in two sets $G = \{y_1, \ldots, y_i\}$ and $H = \{y_{i+1}, \ldots, y_{|Y|}\}$. Given G and H, the desired subsets Y_g and Y_h are derived by

$$Y_g = \{y | y \in G \wedge \nexists \widetilde{y} \in H : (y,\widetilde{y}) \in E_y\} \tag{13}$$
$$Y_h = \{y | y \in H \wedge \nexists \widetilde{y} \in G : (y,\widetilde{y}) \in E_y\} \tag{14}$$

and returned by the *divide* function.

Algorithm 2 order(Y) - Ordering the nodes of the dependency graph.

Require: Y
Ensure: P is an ordered set
1: $D(Y, E_y) :=$ createDependencyGraph(Y)
2: **while** $P \neq Y$ **do**
3: Select $y \in Y \setminus P$ with $\max(w(y,P))$
4: $P := P \cup \{y\}$
5: **end while**
6: **return** P

5. SAT-ASSISTED SIMULATION

This section proposes a novel *SAT-assisted Monte-Carlo simulation* technique. With growing system complexity, analytical methods may become impracticable or even unusable because the final BDDs exhaust available memory as well. An alternative category of reliability analysis techniques that allow to target more complex systems are based on simulation, i.e., *Monte-Carlo simulation*. Simulation has the drawbacks of accurateness being related to the number of performable simulation runs. On the other hand, the memory needed by the introduced formal methods can be traded for runtime and, thus, the problem of outsized BDDs is avoided. By a state-of-the-art SAT-solver based on the DPLL backtracking algorithm [4], a very compact data structure, i.e., a *Conjunctive Normal Form* (CNF) is tested for satisfiability to determine the current system state whenever needed by the simulation. This enables to carry out hundreds of simulation runs in a very short time, even for large and complex systems where known exact methods fail. If the system function is not directly given in CNF like in [8, 6], several efficient techniques are known to transform any Boolean function into CNF, cf. [18].

The iterative SAT-assisted Monte-Carlo simulation approach works as follows: First, a set Γ_φ of N times-to-failure is determined for the overall system encoded in $\varphi(\boldsymbol{x})$ by

$$\Gamma_\varphi = \bigcup_{i=0}^{N} \text{mcs}(\varphi(\boldsymbol{x})) \tag{15}$$

Algorithm 3 mcs($\varphi(\boldsymbol{x})$) - SAT-assisted Monte-Carlo simulation.

Require: $\varphi(\boldsymbol{x})$
Ensure: Γ is an ordered set
1: $\Gamma :=$ timesToFailure(X)
2: **for** $(x, \gamma) \in \Gamma$ **do**
3: $\quad\varphi(\boldsymbol{x}) := \varphi(\boldsymbol{x}) \wedge \neg x$
4: $\quad$**if** $\neg$ sat($\varphi(\boldsymbol{x})$) **then**
5: $\quad\quad$**return** γ // time to failure
6: $\quad$**end if**
7: **end for**

The function mcs : $\{0,1\}^{\{0,1\}^n} \rightarrow \mathbb{R}^+$ carries out one simulation run based on a given structure function $\varphi(\boldsymbol{x})$. The function *mcs* is outlined in Alg. 3. The algorithm first computes a set Γ of times-to-failure in ascending order that contains a specific time to failure γ for each component $x \in X$ of the structure function using the function timesToFailure : $2^X \rightarrow 2^{(X \times \mathbb{R}^+)}$, cf. line 1. The time-to-failure of each system component x is determined by using inverse transform sampling based on the reliability function $\mathcal{R}_x(\tau)$ of the component:

$$\gamma = \mathcal{R}_x^{-1}(r) \tag{16}$$

with $r \in \mathbb{R}_{[0,1]}$ being a random number. For each element $(x, \tau) \in \Gamma$ and with respect to the order of Γ, the structure function φ is combined with a negated component variable $\neg x$ using conjunction, cf. line 3. This corresponds to the component x being failed. The SAT-solver is invoked using the function sat : $\{0,1\}^{\{0,1\}^n} \rightarrow \{0,1\}$ that returns *true* if the structure function can be satisfied, i.e., if the overall system works properly. If the overall system failed, cf. line 4, the time-to-failure of the component that failed last corresponds to the overall system time-to-failure and is returned in line 5.

Given the times-to-failure Γ_φ, the desired reliability function of the system as given in Eq. (1), is approximated as follows:

$$\mathcal{R}(\tau) \approx \frac{|\{\gamma | \gamma \in \Gamma \wedge \gamma > \tau\}|}{N} \tag{17}$$

6. EXPERIMENTAL RESULTS

To give evidence of the effectiveness of the proposed approaches, a comparison to state-of-the-art reliability analysis approaches is given: (1) The proposed early quantification method (EQ) and the SAT-assisted simulation approach (SAT) are compared to a reliability analysis without early quantification like [5] (COMMON) and the partitioning technique presented in [6] (GLRHT08). As described in Sec. 4.1, GLRHT08 corresponds to the preprocessing proposed in the work at hand and is compared based on this preprocessing in order to be independent of the system model used in [6]. (2) The SAT-assisted Monte-Carlo simulation is compared to the reliability analysis presented in [8] that does not use temporary variables such that a comparison to early quantification techniques is impossible. For comparison, a similar algorithm (IH08*) is used that is based on BDDs instead of the so-called *TPDDs* proposed in [8] that, however, leads to a data structure of comparable size and complexity.

Testsuite. For the comparison of the proposed and state-of-the-art approaches, a testsuite containing various system-level design specifications is arranged: The testsuite contains both, real-world as well as synthetic test cases. The real-world examples exhibit a certain structure in both application and architecture that is the result of the structured development process. This structure often allows for a better partitioning and an easier analysis. On the other hand, the synthetic examples lack that certain structure because these examples are randomly generated. This randomness commonly makes analysis harder and often leads to significantly larger data structures when compared to structured test cases of equal size. 7 real-world specifications from the data-streaming as well as the automotive domain are chosen. The complexity of the real-world test cases ranges from about 50 tasks with 30 available resources up to about 250 tasks with about 1000 available resources. Moreover, 8 synthetic test cases are generated. The complexity of the synthetic test cases ranges from 50 tasks with 25 available resources to 150 tasks with 75 available resources. For each of the 15 test cases, 10 implementations of different complexities with respect to the BDD sizes are generated: Using very few resources with marginal task redundancy creates implementations of low complexity, whereas using many resources with a high amount of task redundancy results in implementations of high complexity. The result is a testsuite of 150 test cases that covers a broad variety of test instances ranging from small examples up to highly-complex real-world test cases that also max out state-of-the-art design space exploration and performance evaluation approaches, cf. [14]. The experiments are carried out on an Intel Pentium 4 3.00 GHz Dual Core machine with 1.5 GB RAM. The number of simulation runs for the SAT approach is set to 2000.

The results for the comparison of SAT, EQ, GLRHT08, and COMMON are depicted in Fig. 4:

Runtime. The scatter plots show that in very few cases, the overhead resulting from proposed early quantification can slightly increase runtime τ_{RT}. However, for more complex systems to analyze, the runtime of EQ is significantly lower than the runtime of both former approaches COMMON and GLRHT08. The runtime of the proposed SAT approach is significantly larger for all test cases where the exact methods were able to deliver feasible results. However, the advantage of SAT lies in its scalability, discussed in the following.

Scalability. Since the paper focuses on the scalability of the proposed methods, the number of test cases, where no feasible analysis was possible due to outsized BDDs, is taken as a measure for scalability. The COMMON approach performs worst and fails in 95 test cases. GLRHT08 performs better, but still fails in 35 cases. The proposed EQ approach fails only in 18 cases and never failed where one of the known approaches succeeded. Thus, the proposed early quantification approach outperforms known approaches on reasonable complex test cases. However, only the proposed SAT approach was able to solve each test case. Note that one test case from the automotive area was one order of magnitude larger with respect to the number of components compared to all other test cases. Thus, the proposed SAT-assisted simulation approach has the best performance in terms of scalability. However, it should be replaced by the proposed early quantification approach whenever possible to take advantage of the lower runtime and exact results of EQ.

Accuracy. Since EQ, GLRHT08, and COMMON are exact approaches with the SAT approach only being an approximation of the reliability function of the system, the relative error in percent is determined based on the 132 test cases where an exact reliability function could be derived. The relative error is determined based on the Mean-Time-To-Failure $MTTF = \int_0^\infty \mathcal{R}(\tau) d\tau$ that is derived from the approximated and exact reliability functions. The relative error for 2000 simulation runs per test case is 1.51% with a standard deviation of 1.02%. For a reliability analysis at system-level, the accuracy can be considered very good, especially with respect to the SAT approach having its main application where known exact methods fail. For comparison, the relative error for 500 test-runs is 4.01% with a standard deviation of 2.08% while the relative error for 4000 runs is 1.18% with a standard deviation of 0.94%.

The results of the comparison of SAT and IH08* can be found in Fig. 5. The memory-consuming BDDs that result from the analysis of both transient and permanent faults allow to highlight the scalability of the SAT approach. While

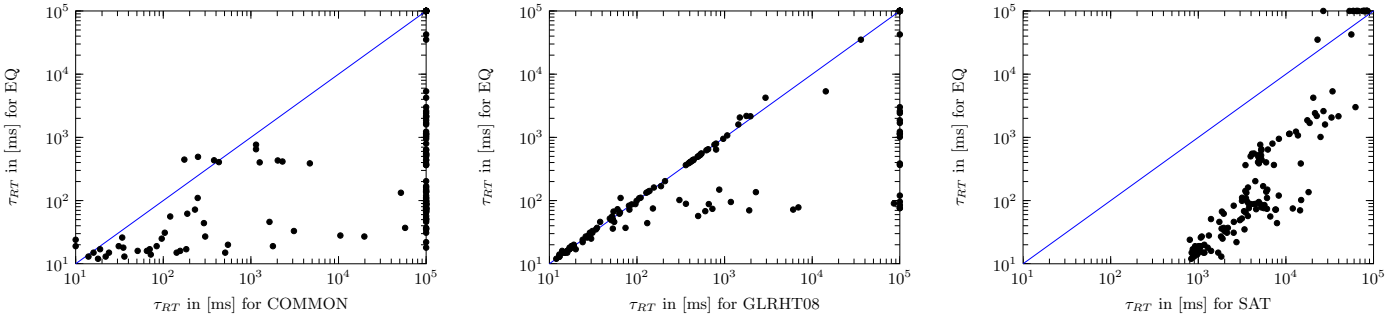


Figure 4: Comparison of the time consumption τ_{RT} of the proposed SAT-assisted simulation (SAT), the proposed early quantification approach (EQ), the partitioning presented in [6] (GLRHT08), and a common analysis without early quantification (COMMON) for 150 selected test cases. Note that test cases where an approach ran out of memory are set to the maximum value on the corresponding axis. Axes are given in loglog-scale.

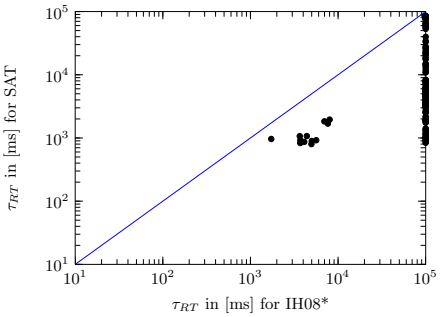


Figure 5: Comparison of the time consumption τ_{RT} of the proposed SAT-assisted simulation (SAT) and the analysis proposed in [8] (IH08*) for 150 selected test cases. Note that test cases where IH08* ran out of memory is set to the maximum value on the x-axis. Axes are given in loglog-scale.

SAT was able to analyze every test case successfully, IH08* was able to deliver feasible results in 12 test cases only and failed for 138 test cases. This shows the ability of the SAT-assisted simulation to increase scalability and its good performance, especially when more sophisticated analysis techniques lead to outsized BDDs already for relatively small test cases.

7. CONCLUSION

This paper proposes two approaches to tackle the problem of memory-exhausting Binary Decision Diagrams (BDDs) in state-of-the-art automatic system-level reliability analysis of embedded systems. The contributions of this paper are (1) a symbolic early quantification technique that keeps the size of the BDDs during construction small and (2) a SAT-assisted simulation approach that allows to deliver appropriate results for large and complex systems where the final BDDs used in exact approaches exhaust available memory. A test-suite of 150 test cases consisting of synthetic examples as well as problem instances from the data-streaming and automotive domain have been used to show the scalability of the proposed approaches. The presented early quantification approach outperforms known exact methods and decreased the number of test cases where the BDDs exhausted available memory by nearly 50% compared to the best known approach. The SAT-assisted simulation was capable of analyzing all given test cases at the costs of an increased runtime. Thus, the presented approaches in the work at hand enable the application of reliability analysis techniques to problems of industrial relevance where known approaches from literature failed due to the problem size.

In the future, other early quantification approaches shall be applied to formal reliability analysis and compared to the proposed approach. Moreover, the applicability of the SAT-assisted simulation approach to take the *repair* of resources into account shall be investigated.

8. REFERENCES

[1] R. Billinton and W. Li. *Reliability assessment of electrical power systems using Monte Carlo methods*. Plenum Publishing Corporation, 1994.

[2] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *Trans. on Computers*, 35(8):677–691, 1986.

[3] D. W. Coit and A. E. Smith. Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm. *IEEE Transactions on Reliability*, 45(1):254–260, 1996.

[4] M. Davis et al. A machine program for theorem-proving. *Comm. of the ACM*, 5(7):394–397, 1962.

[5] M. Glaß et al. Reliability-Aware System Synthesis. In *Proc. of DATE '07*, pages 409–414, 2007.

[6] M. Glaß et al. Symbolic Reliability Analysis and Optimization of ECU Networks. In *Proc. of DATE '08*, pages 158–163, 2008.

[7] R. Hojati et al. Early quantification and partitioned transition relations. In *Proceedsing of ICCD '96*, pages 12–19, 1996.

[8] A. Israr and S. Huss. Specification and design considerations for reliable embedded systems. In *Proc. DATE '08*, pages 1111–1116, 2008.

[9] A. Israr et al. An efficient reliability evaluation approach for system-level design of embedded systems. In *Proc. of the 2009 10th ISQED '09*, pages 339–344, 2009.

[10] V. Izosimov et al. Synthesis of fault-tolerant schedules with transparency/performance trade-offs for distributed embedded systems. In *Proc. of DATE '06*, pages 706–711, 2006.

[11] V. Izosimov et al. Analysis and Optimization of Fault-Tolerant Embedded Systems with Hardened Processors. In *Proc. DATE '09*, pages 682–687, 2009.

[12] J. Jain et al. Decomposition Techniques for Efficient ROBDD Construction. In *Proceedings of FMCAD '96'*, pages 419–434, 1996.

[13] A. Jhumka et al. A dependability-driven system-level design approach for embedded systems. In *Proc. of DATE '05*, pages 372–377, 2005.

[14] M. Lukasiewycz et al. Combined System Synthesis and Communication Architecture Exploration for MPSoCs. In *Proceedings of DATE '09*, pages 472–477, 2009.

[15] R. Ranjan et al. Efficient BDD Algorithms for FSM Synthesis and Verification. In *Proceedings of IWLS '95*, 1995.

[16] A. Rauzy. New Algorithms for Fault Tree Analysis. *Reliability Eng. and System Safety*, 40:202–211, 1993.

[17] S. Tosun et al. Reliability-Centric High-Level Synthesis. In *Proc. of DATE '05*, pages 1258–1263, 2005.

[18] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic*, 2(115-125):10–13, 1968.

[19] S. Weaver et al. Extending Existential Quantification in Conjunctions of BDDs. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:89–110, 2006.

[20] Y. Xie et al. Reliability-Aware Cosynthesis for Embedded Systems. In *Proc. of ASAP '04*, pages 41–50, 2004.

[21] Y. Zhang et al. Energy-aware deterministic fault tolerance in distributed real-time embedded systems. In *Proc. of DATE '05*, pages 372–377, 2005.

[22] C. Zhu et al. Reliable multiprocessor system-on-chip synthesis. In *Proc. of CODES/ISSS '07*, pages 239–244, 2007.