Cone of Influence Analysis at the Electronic System Level Using Machine Learning

Jannis Stoppe† Robert Wille*† Rolf Drechsler*†
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{stoppe,rwille,drechsle}@informatik.uni-bremen.de

***Abstract*—Cone of influence analysis, i.e. determining the parts of the circuit which are relevant to a considered circuit signal, is an established methodology applied in several design tasks. In abstractions like the *Register Transfer Level* (RTL) or the gate level, cone of influence analysis is simple. However, the introduction of higher levels of abstractions, particularly the *Electronic System Level* (ESL), made it significantly harder to reliably extract a cone of influence.**

In this paper, we propose a methodology that enables cone of influence analysis at the ESL. Instead of a structural analysis, a behavioral scheme is proposed, i.e. stimuli representing different system executions are analyzed. To this end, machine learning techniques are exploited. This enables a very good approximation of the desired cone of influence which is non-invasive, does not rely on the availability of the source code, and performs fast. Case studies confirm the applicability of the proposed approach.

I. Introduction

Knowing the dependencies within hardware systems has always been a key issue in several design tasks. As a prominent example, cone of influence analysis is an established method which is heavily applied e.g. in design understanding [8], debugging [1], verification [3], [4], [6], and more. The general idea is thereby to take only those parts of the circuit into consideration that are relevant to the respective design task. As long as circuits and systems are designed and verified at the *Register Transfer Level* (RTL) or the gate level, the extraction of the cone of influence is simple.

However, due to the increasing complexity of circuits and systems, designers strive for higher levels of abstraction. This eventually led to the design at the *Electronic System Level* (ESL) [2] with *SystemC* [15] being its de-facto standard [13], [18]. Here, the desired system is no longer implemented through a netlist description which is composed of signals connected to components or gates. Instead, the system is implemented in an algorithmic fashion from which, eventually, a binary to be executed is derived. As a consequence, the "classical" structure of a hardware system is no longer available leading to crucial obstacles for cone of influence extraction.

In fact, SystemC incorporates the full expressive power of C++ and, by this, allows for a significant amount of different description means. Compared to the simple netlist, this significantly complicates a cone of influence analyzer following the "classical" scheme known from the RTL and gate level. Moreover, at the ESL, the entire source code of the considered system is not always available. As soon as pre-compiled libraries, user inputs, multiple threads, or network access are used, cone of influence determination cannot rely on the actual netlist or code anymore. Finally, side effects harden the analysis further. For example, if a (partial) cone of influence includes a global variable, the number of further components possibly also being part of the cone of influence increases significantly. Because of reasons like this, cone of influence analysis became significantly harder at the ESL.

In this work, we present an approach which addresses this problem. Instead of a structural analysis as commonly applied at RTL and gate level, a behavioral scheme is proposed. That is, stimuli representing various executions of the system under consideration are analyzed. From the results, eventually, the desired information on the cone of influence for a considered signal are derived. To this end, methods from the domain of *machine learning* [17] are exploited.

Following this scheme leads to an approximation of the cone of influence, i.e. exactness of the results is not guaranteed. However, the proposed approach offers a promising alternative to the "classical" cone of influence analysis which provides

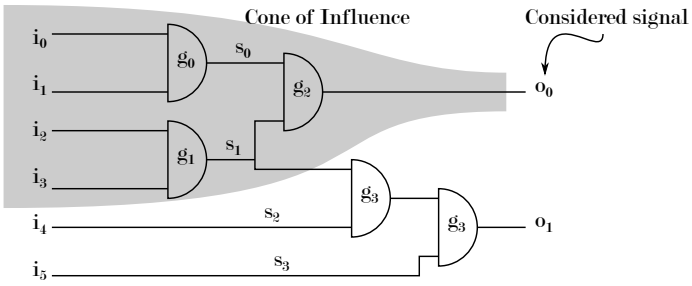
1) a non-invasive methodology, i.e. no changes in the code have to be performed,
2) a behavioral-based methodology that does not rely on the source code of a system, but only on its simulated behavior, and
3) a fast methodology, since the respective analyses can be performed in negligible run-time.

Furthermore, case studies confirm that the quality of the achieved approximations ranges from very good to sufficient. In the majority of the cases, in fact, the *exact* cone of influence can be obtained. But also if only an approximation returns, the differences to the actual cone of influence remain small.

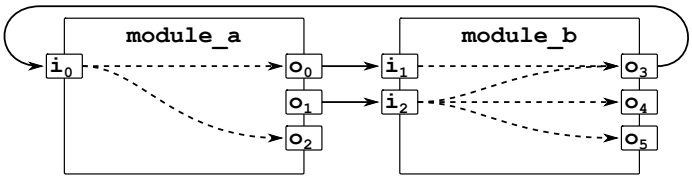
In the remainder of this paper, the proposed methodology is described in detail. First, the problem is precisely defined in Section II. Afterwards, the proposed approach is described and discussed in detail in Section III. Section IV summarizes the results obtained by the conducted case study. Finally, Section V concludes the paper.

II. Problem Formulation

Cone of influence analysis is an established methodology in the design of circuits and systems. The general idea is to take only those parts of the circuit into consideration that are relevant to the respective design task. Applications can e.g. be found in the following domains:



(a) At the RTL or gate level



(b) At the ESL

Fig. 1. Cone of influence analysis

- Design understanding [8]: If a designer wants to understand the purpose of a signal, only the components triggering this signal need to be inspected.
- Debugging [1]: If an erroneous behavior occurs on a signal, the reason for this error must be within the components triggering this signal.
- Verification [3], [4], [6]: If the correctness of a signal shall be verified, the verification engine only has to consider the components triggering this signal.

At abstractions like RTL or gate level, a cone of influence of a signal s in a circuit is the set of all signals as well as modules and gates the considered signal s depends on. Here, these components can easily be obtained by a backward traversal starting from the currently considered signal.

Example 1. *Consider the gate level circuit shown in Fig. 1(a) and assume the cone of influence for the output signal o_0 shall be determined. Since o_0 depends on gate g_2 and its two inputs s_0 and s_1, these elements are added in the cone of influence. Since, in turn, s_0 and s_1 depend on gates g_0 and g_1 with their inputs i_0, i_1, i_2, and i_3, also these components are added. In contrast, o_0 does neither depend on the gates g_3 and g_4 nor on their input signals. Hence, while the overall circuit is composed of five gates in total, only three of them need to be considered in order to observe o_0.*

While the cone of influence analysis is trivial for circuits provided at the RT level or the gate level, new obstacles are introduced if a circuit or system is available at the ESL. Particularly if SystemC, the de-facto standard in ESL, is applied, problems occur. Here, after the compilation, the resulting binary is stripped of all meta information which leaves the designer without proper analyzing capabilities. Moreover, even if the source code is still available, a sufficient cone of influence analysis is often not possible due to the following reasons:

- SystemC allows for a significant amount of different description means (the full expressive power of C++ including its dialects) which need to be supported by the respective analyzer. This, of course, can be addressed by altering the code so that it is composed of supported constructs only. However, this might become a time-consuming and, therefore, expensive task.
- The availability of source code usually stops at pre-compiled libraries, user inputs, multiple threads, file or network access, and all other possibilities of the program interfacing with any other element other than source code. For all these elements, it is unknown how to obtain the respective cone of influence.
- Global variables could be changed by any function or library call at any time. Consequently, if a (partial) cone of influence includes a global variable, the number of further components possibly also being part of the cone of influence increases significantly. This results in the accumulation of dependencies that might not be part of the logic behind the source code.
- In order to reduce these overestimations, an analyzer needs to decide whether it should act conservatively or progressively, i.e. either taking all possible value changes into account (and maybe end up discovering much more dependencies than there are) or leaving out those that it deems reasonable (and maybe end up not having the one connection that is important for a certain case).

As a result, if source code is not or only partially available (which might already be the case when using standard library calls), the cone of influence analysis at ESL is restricted as illustrated by means of Fig. 1(b): If available, the respective connections between SystemC modules can be extracted via an API (solid lines). In contrast, connections within a module (dashed lines) cannot be derived if the implementation of these modules is not available, if the corresponding description means are not supported, or if their respective values might be affected through side effects like global variables. Consequently, e.g. the complete cone of influence of signal o_0 of *module_b* cannot exactly be determined yet.

Motivated by this, in the remainder of this paper we propose an alternative to the "classical" approach of cone of influence extraction which supports the new requirements at the ESL.

III. Proposed Approach

In this section, the proposed approach for cone of influence analysis at the ESL is presented. First, the general idea is outlined before the respective steps are described in detail.

A. General Idea

The "classical" approach for cone of influence analysis suffers from the obstacles discussed above when being applied to ESL designs. Hence, instead of a structural analysis, a behavioral scheme is proposed. To this end, a set of stimuli is applied to the system with the intent of triggering various behaviours. Afterwards, the relations between the respective signal assignments are observed in order to deduce the desired information. The following example illustrates the idea.

Example 2. *Consider the abstract SystemC module as illustrated in Fig. 2 and assume that the cone of influence for the output signal o_0 shall be determined by means of a behavioral analysis. To this end, 19 input/output assignments are provided. At a first glance, these assignments are merely detached streams of Boolean values. However, at a second glance, certain characteristic relations can be observed. For example, each time input i_3 is set to 0, the considered signal o_0 outputs 1. This may lead to the conclusion that the value of o_0 depends on the value of i_3. In a similar fashion, further conclusions can be drawn.*

Obviously, relations like the one discussed in this example are unintuitive and hard to grasp for human beings. Furthermore, the conducted implications might be premature and misleading, i.e. wrong conclusions may be drawn. However, considering a significant amount of stimuli and applying a comprehensive analysis of such relations, quite reliable information can be extracted. In fact, analyzing such relations and deducing information from that has intensely been considered in the past – forming the domain of *machine learning* [17].

In the following, a state-of-the-art technique of machine learning is exploited to determine the desired cone of influence. Following this idea has the following advantages:

- It is *non-invasive*. As only the interfaces of a module are utilized, the source code does not need to be adjusted or changed – regardless of the existing code base. By this, the proposed approach also does not rely on compiler-specific additions and changes.
- Its results are based on *observed correlations* instead of potentially unused connections in the source code. If, for example, the output signal is defined by $a \wedge b$, a structural analysis will assume that it depends on both inputs, a and b. If, however, b can only be true if a is true as well, b in fact does not actually affect the output. This is especially important if the source code for a calculation is not available.
- It is *fast*. Although the process of finding the optimal signal to divide a given set based on the information gain itself is complex, the data can be handled efficiently.

Due to these reasons and despite the disadvantages resulting from its indeterminism, applying machine learning offers a promising alternative for an efficient cone of influence extraction at the ESL. To this end, two major steps are needed to be performed: First, a sufficient set of stimuli to be analyzed is created. Afterwards, this set is analyzed through machine learning leading to the desired results. In the following, both steps are described in detail.

B. Stimuli Generation

The proposed approach determines the desired cone of influence from a thorough analysis of certain assignments to the signals of the considered system from which characteristics of their relations are automatically deduced. Consequently, the quality of the approach significantly relies on the set of stimuli applied to the system. For the machine learning algorithm to perform well, a number of "good" stimuli to be considered need to be generated. To this end, different strategies may solely or in parallel be applied (see e.g. [23]). They should particularly incorporate the following two characteristics:

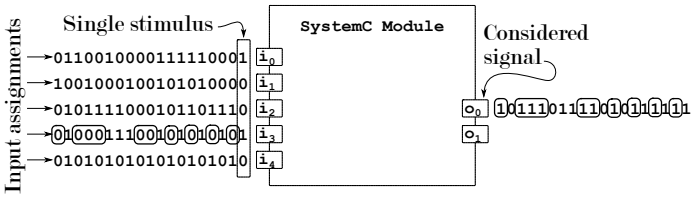


Fig. 2. Illustrating the general idea

1) Diversity: An important indicator for a well-chosen set of stimuli is the diversity of the assignments to the considered signal that is supposed to be classified. If these assignments rarely change, the machine learning algorithm will usually have trouble drawing helpful conclusions. For example, consider a simple system computing $c = a \wedge b$. If only stimuli $\{a = 0, b = 0\}, \{a = 0, b = 1\}, and \{a = 1, b = 0\}$ are applied, a possible conclusion could be that signal c is *always* set to 0. In order to avoid that, stimuli leading to $c = 1$ should also be considered. Following the same reasoning, also the assignments to the respective input signals should be diverse. Otherwise, it is harder to determine which input signals actually triggered a change in the signal considered for cone of influence analysis.

In general, the machine learning algorithm attempts to "tidy up" a given set of cluttered signal assignments and, by doing this, extracts the desired information. Therefore, the provided set of data should be as "untidy" or as diverse as possible. For the considered signal, this is not easy to achieve since its values are determined by the module's inner, unknown structure. However, many approaches for stimuli generation have been proposed in the past, which can be exploited to satisfy these requirements including approaches for simple random simulation or directed simulation [23], or more elaborated methods like e.g. constraint-based random simulation [21], [22].

2) Quantity: Apart from the assignments to be applied to the signals, the amount of different assignments is also an important factor to the quality of the result. In general, the more stimuli are generated, the better results can be achieved. However, when using machine learning algorithms to classify a given signal, the algorithms tend to suffer from *overfitting* when they are exposed to too much information (see e.g. [10]). But when deterministic behavior is considered (as it is the case for systems specified in ESL), the applied machine learning approaches are usually hardly affected by this.

Example 3. *Consider again the abstract SystemC module as illustrated in Fig. 2. The depicted assignments already satisfy the characteristics discussed above very well. The assignments to both, the considered signal as well as the inputs, are rather diverse. Furthermore, an adequate number of stimuli is available. As shown in the next section, this set leads to very precise conclusions on the cone of influence.*

C. Machine Learning

Using the assignments generated in the first step, relations between them from which the desired information can be deduced are determined next. To this end, a machine learning approach is utilized. Basically, the idea of machine learning is to locate those patterns in the given set that provide the simplest explanation possible of the given phenomenon. This follows Occam's razor which states that, if there are several theories that explain a given phenomenon, the one making the least assumptions probably is the right one [10]. This idea of mere correlation implying causation usually should be applied carefully. However, especially in discrete and manageable environments like system designs, the assumption that correlating signals are somehow connected certainly is legitimate.

While there are many algorithms that can be used to classify data sets (see e.g. [11]), we applied the C4.5 algorithm introduced in [16] for the purpose of cone of influence analysis. This approach is widely used and has been proven efficient in many applications as computer vision [7], language processing [12], medical diagnosis [24], financial analysis [14], general game playing [19], robotics [5] and others.

C4.5 takes a given data set and recursively splits this set into more "tidy" sub-sets. To this end, all possible splits are previously evaluated by means of the entropy function $-(p_0 \log_2(p_0) + p_1 \log_2(p_1))$, where p_0 (p_1) denotes the probability of the considered signal being set to 0 (1) according to the data set. If no progress can be obtained from further splitting anymore, the algorithm terminates. Then, from the resulting tree, conclusions concerning the cone of influence can be drawn. More precisely, the following steps are performed:

1) Determine the entropy of the currently considered set.
2) Determine an input signal on which the currently considered data set shall be split. To this end, apply the entropy function, i.e. chose the input signal which would lead to two sub-sets whose entropy (i.e. "tidiness") is better than in the currently considered set.
3) In case no such input signal could be determined, add a leaf node to the decision tree and terminate. This happens if all available stimuli (i.e. the given data set)
   - always lead to a constant assignment for the considered signal or
   - cannot further be refined through splitting.
4) Split the currently considered data set with respect to the chosen signal. Afterwards create a decision node whose successors pin-point to the newly created sub-sets.
5) Recursively start over at Step 1 using the newly created sub-sets.

From the resulting decision tree, the cone of influence of the considered signal can be deduced. In fact, each decision node actually represents an input signal which mostly refined the values of the considered signal. From this, it can be concluded that the respectively chosen signal actually influences the considered signal and, hence, is likely be a part of the cone of influence.

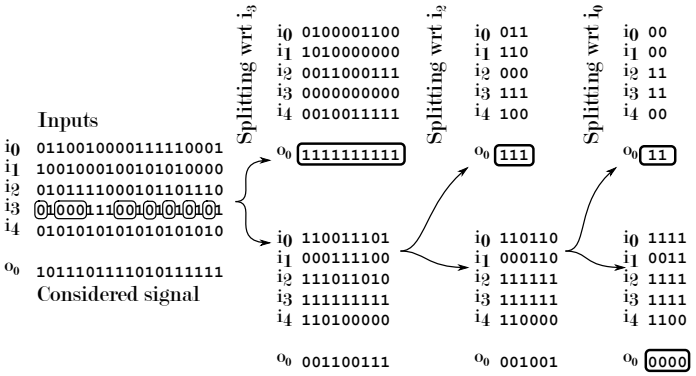


Fig. 3. Applying machine learning

Example 4. *The described scheme is applied to the example from Fig. 2. First, the entropy of the given set is calculated. To this end, the entropy function* $-(p_0 \log_2(p_0) + p_1 \log_2(p_1))$ *is applied. That is, using all stimuli from Fig. 2, an entropy of* $-\frac{4}{19} log_2(\frac{4}{19}) - \frac{15}{19} log_2(\frac{15}{19}) \approx 0,742$ *results. Based on this, it is determined which splitting on what signal would lead to sub-sets with the best entropy. For example, if the data set from Fig. 2 would be split with respect to input* i_0*, a sub-set of 10 stimuli and entropy of 0 (considered signal is always* 0*) and a sub-set of 9 stimuli and entropy of* 0.991 *would result leading to an average entropy of* $\frac{10}{19} \cdot 0 + \frac{9}{19} \cdot 0.991 = 0.469$*. Overall, splitting with respect to the available inputs would lead to the following average entropies:*

- i_0*:* $\frac{10}{19} \cdot 0 + \frac{9}{19} \cdot 0,991 = 0,469$
- i_1*:* $\frac{12}{19} \cdot 0.619 + \frac{6}{19} \cdot 0.918 = 0.681$
- i_2*:* $\frac{9}{19} \cdot 0.619 + \frac{11}{19} \cdot 0.946 = 0.547$
- i_3*:* $\frac{10}{19} \cdot 0 + \frac{9}{19} \cdot 0.991 = 0.469$
- i_4*:* $\frac{10}{19} \cdot 0.722 + \frac{9}{19} \cdot 0.764 = 0.742$

Hence, splitting with respect to i_0 *and* i_3 *would lead to the most tidy sub-sets. Assume the algorithm decides for* i_3*. Then, a decision node and two new sub-sets as shown in Fig. 3 result. Here, it can be seen that the first sub-set is only composed of stimuli setting the considered signal to* 1*, i.e. the entropy is* 0 *and this sub-set is not further split. In contrast, the second sub-set can further be refined. This is done in the remaining iterations eventually leading to the complete decision tree as shown in Fig. 3.*

From this decision tree, it can now be deduced that the considered signal very likely depends on the inputs i_0*,* i_2*, and* i_3*, i.e. the input signals that had the most possible effect to it. In contrast, inputs* i_1 *and* i_4 *are not part of this tree and, hence, probably do not influence the considered signal.*

IV. Case Studies

As discussed in Section III-A, the proposed approach approximates the actual cone of influence. To determine how far the approximation differs from the actual cone of influence, i.e. to evaluate the quality of the approach, several case studies have been conducted. To this end, the proposed approach has been implemented in C++. For stimuli generation, a random scheme has been applied. For machine learning, the C4.5

implementation provided by the *WEKA framework* (J48) [9] was utilized. All case studies have been conducted on a AMD Phenom II X4 machine with 3.4 GHz and 8 GB of memory running Windows 7. In this section, the results of the case studies are summarized and discussed.

A. Considered Benchmarks

In order to ensure a precise analysis, we evaluated the proposed approach using specifically generated SystemC models that realize arbitrary logic operations. More precisely, SystemC programs have been generated that instantiated a module with n inputs and a single output (representing the considered signal for which a cone of influence shall be determined). The output value was thereby triggered by a randomly generated functional polynom based on an arbitrarily chosen set of inputs. By this, we were able to easily track the exact cone of influence (which simply was constituted by the applied monoms of the polynom), while, at the same time, providing a realistic scenario in which the proposed approach can be evaluated.

B. Results

Table I summarizes the results obtained in the case studies. The first two columns provide the number of inputs of the considered SystemC modules as well as the number of their operations. Afterwards, the results of the proposed cone of influence analysis are presented which have been obtained when either 10, 20, 50, 100, 200, 500, 1,000, 5,000, or 10,000 stimuli were applied. Column ✓ respectively denotes thereby the number of input signals which have correctly been identified as being part of the cone of influence. Column f_p respectively denotes the number of incorrectly classified input signals (i.e. the *false positives*), while column f_n respectively denotes the number of missed input signals (i.e. the *false negatives*). That is, in all cases with $f_p = 0$ and $f_n = 0$, the exact cone of influence has been determined. In all other cases, either too many (if $f_p > 0$) and/or too few (if $f_n > 0$) input signals have been classified to be in the cone of influence, i.e. either an over-approximation and/or and under-approximation resulted. The last two columns provide the number of incorrectly classified input signals ($\frac{f_p}{Inp.}$) and missed input signals ($\frac{f_n}{Inp.}$) as a percentage of the total number of inputs. *All* results reported in Table I have been obtained in less than one CPU minute, i.e. in negligible run-time.

The results confirm the discussions from the previous sections. The following conclusions can be drawn:

- Applying machine learning indeed enables an efficient cone of influence approximation for system descriptions at the ESL. All results have been determined in negligible run-time.
- The quality of the approximations ranges from very good to sufficient. In fact, in more than two Thirds of the cases, the *exact* cone of influence was determined (all entries with $f_n = 0$ and $f_p = 0$).
- In the remaining cases, the results should be distinguished between false positives and false negatives. A large number of false positives ($f_p \gg 0$) represents an over-approximation, i.e. more signals than necessary are considered. This is unwanted, but not crucial. Much more relevant is a large number of false negatives ($f_n \gg 0$) as they represent missed signals, i.e. signals which are entirely not considered although they influence the considered signal. As can be seen in Table I, in the few cases where no exact result has been achieved, this number of false negatives usually is small. That is, even if it was not possible to determine an exact result, a sufficient approximation was obtained.
- Finally, the effect of the number of applied stimuli on the quality of the approximations can be observed. The more stimuli are applied, the closer the approximated cone on influence is to the exact one. This particularly holds for larger designs which, obviously, require more stimuli to get better approximations.

Overall, the proposed approach provides good approximations, in many cases even the exact determination, of the desired cone of influence in SystemC designs.

In order to test the behaviour of the algorithm if some of the variables remain hidden (e.g. because they are internal states of a library and cannot be logged), the same test runs were also executed with $min(\frac{inp}{5} \cdot 2, \frac{op}{3} \cdot 2)$ of the used variables remaining hidden to the ML algorithm. While the results suffer from an increasing number of false positives as the amount of applied stimuli increases, the false negatives remain unaffected (according to the Wilcoxon signed-rank test [20]). This complies with the aim to keep the amount of missed signals low.

V. Conclusion

In this work, we presented a methodology for cone of influence analysis at the ESL. For this purpose, techniques for machine learning have been exploited. The resulting approach is non-invasive, does not rely on the availability of the code, and performs fast, but does not guarantee exactness of the results. However, case studies confirmed that, in many cases, the exact cone of influence can be achieved. Moreover, even if this was not possible, a sufficient approximation could be obtained. The method could be applied for design understanding, speeding up tests and verification tasks and visualizing an approximation of a module's interior structure.

For future work, a more detailed evaluation of the available machine learning methods is left. Furthermore, dedicated stimuli generation for the sake of cone of influence analysis seem to be a promising direction which may help to further improve the reported results.

VI. Acknowledgments

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project *VisES* under contract no. 16M3197B and the German Research Foundation (DFG) within the Reinhart Koselleck project under contract no. DR 287/23-1.

TABLE I
EXPERIMENTAL EVALUATION

		Results for a Number k of Applied Stimuli																												Inaccuracy wrt. Inputs	
		k=10			k=20			k=50			k=100			k=200			k=500			k=1,000			k=5,000			k=10,000				
Inp.	Op.	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	✓	f_p	f_n	$\frac{f_p}{Inp.}$	$\frac{f_n}{Inp.}$
5	3	1	1	1	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	0%	0%
	6	0	0	3	0	0	3	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	0%	0%
	15	1	0	3	2	0	2	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	0%	0%
	30	0	0	5	4	0	1	3	0	2	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	0%	0%
	60	0	0	5	3	0	2	4	0	1	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	0%	0%
10	3	1	0	1	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	0%	0%
	6	1	1	3	2	2	2	4	1	0	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	4	0	0	0%	0%
	15	1	0	7	3	0	5	5	0	3	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0	0%	0%
	30	2	0	7	2	0	7	5	1	4	7	1	2	7	0	2	9	1	0	9	1	0	9	1	0	9	1	0	10%	0%
	60	0	0	10	0	0	10	5	0	5	0	0	10	10	0	0	8	0	2	10	0	0	10	0	0	10	0	0	0%	0%
20	3	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0	0%	0%
	6	0	1	5	1	1	4	5	0	0	5	1	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	0%	0%
	15	1	0	9	2	2	8	4	0	6	9	3	1	9	6	1	10	4	0	10	4	0	10	0	0	10	0	0	0%	0%
	30	1	0	15	3	0	13	6	0	10	11	2	5	12	3	4	16	3	0	16	4	0	16	4	0	16	4	0	20%	0%
	60	2	0	18	3	0	17	3	0	17	5	0	15	11	0	9	13	0	7	17	0	3	17	0	3	17	0	3	0%	15%
50	3	0	0	3	0	0	3	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	0%	0%
	6	0	0	5	2	0	3	3	2	2	5	4	0	5	0	0	5	0	0	5	0	0	5	0	0	5	0	0	0%	0%
	15	1	0	13	2	0	12	3	3	11	8	11	6	10	8	4	13	23	1	14	25	0	14	23	0	14	18	0	36%	0%
	30	1	0	22	2	0	21	5	2	18	7	3	16	8	8	15	16	9	7	16	13	7	16	17	7	16	17	7	34%	14%
	60	0	0	36	0	0	36	2	1	34	3	2	33	6	5	30	11	6	25	19	5	17	22	9	14	22	9	14	18%	28%
100	3	0	0	3	0	0	3	1	2	2	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	0%	0%
	6	0	1	6	1	2	5	2	5	4	6	2	0	6	2	0	6	0	0	6	0	0	6	0	0	6	0	0	0%	0%
	15	1	0	13	0	2	14	1	2	13	4	6	10	9	15	5	13	24	1	13	30	1	13	34	1	13	29	1	29%	1%
	30	0	1	26	1	1	25	2	6	24	5	3	21	9	14	17	17	18	9	18	35	8	21	59	5	21	60	5	60%	5%
	60	0	1	48	2	0	46	3	1	45	7	0	41	8	8	40	15	12	33	21	15	27	39	31	9	40	38	8	38%	8%

Inp.: Number of inputs Op.: Number of operations ✓: Number of correctly classified input signals

f_p: Number of incorrectly classified input signals (i.e. *false positives*) f_n: Number of missed input signals (i.e. *false negatives*)

Inaccuracy wrt. Inputs: Incorrectly classified input signals ($\frac{f_p}{Inp.}$) and missed input signals ($\frac{f_n}{Inp.}$) as a percentage of the total number of inputs

All results have been determined in less than one CPU minute.

REFERENCES

[1] Moayad Fahim Ali, Sean Safarpour, Andreas G. Veneris, Magdy S. Abadir, and Rolf Drechsler. Post-verification debugging of hierarchical designs. In *ICCAD*, pages 871–876, 2005.

[2] Brian Bailey. *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Elsevier, San Diego, CA, 2007.

[3] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In Willem-Paul Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *Lecture Notes in Computer Science*, pages 81–102. Springer Berlin Heidelberg, 1998.

[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. pages 193–207, March 1999.

[5] James Brusey and Lin Padgham. Techniques for obtaining robust, real-time, colour-based vision for robotics. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, volume 1856 of *Lecture Notes in Computer Science*, pages 63–73. Springer Berlin / Heidelberg, 2000.

[6] Alessandro Cimatti, Enrico Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Integrating bdd-based and sat-based symbolic model checking. In Alessandro Armando, editor, *Frontiers of Combining Systems*, volume 2309 of *Lecture Notes in Computer Science*, pages 265–276. Springer Berlin / Heidelberg, 2002.

[7] Mark A. Friedl, Douglas K. McIver, John C.F. Hodges, X.Y. Zhang, D. Muchoney, Alan H. Strahler, Curtis E. Woodcock, Sucharita Gopal, Annemarie Schneider, Amanda Cooper, Alessandro Baccini, Feng. Gao, and Crystal Barker Schaaf. Global land cover mapping from modis: algorithms and early results. *Remote Sensing of Environment*, 83(1-2):287–302, 2002.

[8] Daniel Große, Rolf Drechsler, Lothar Linhard, and Gerhard Angst. Efficient automatic visualization of systemc designs. In *Forum on Specification & Design Languages*, pages 646–658, 2003.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[10] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 35(19):1–12, 2004.

[11] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[12] Inderjeet Mani and George Wilson. Robust temporal processing of news. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 69–76, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[13] Kevin Marquet, Matthieu Moy, and Bageshri Karkare. A theoretical and experimental review of SystemC front-ends. In *Forum on Specification and Design Languages*, pages 124–129, 2010.

[14] Edmar Martinelli, André de Carvalho, Solange Rezende, and Alberto Matias. Rules extractions from banks' bankrupt data using connectionist and symbolic learning algorithms. In *Proceedings of the Sixth International Conference on Computational Finance*, pages 515–533, 1999.

[15] O.S.C. Initiative. IEEE Standard SystemC Language Reference Manual. *IEEE Computer Society*, 2006.

[16] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[17] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[18] Carsten Schulz-Key, Markus Winterholer, Thomas Schweizer, Tommy Kuhn, and Wolfgang Rosentiel. Object-oriented modeling and synthesis of SystemC specifications. In *Asia and South Pacific Design Automation Conference*, pages 238–243, 2004.

[19] Xinxin Sheng and David Thuente. Using decision trees for state evaluation in general game playing. *KI - Künstliche Intelligenz*, 25:53–56, 2011.

[20] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[21] Robert Wille, Daniel Große, Finn Haedicke, and Rolf Drechsler. Smt-based stimuli generation in the systemc verification library. In *FDL*, pages 1–6, 2009.

[22] Jun Yuan, A. Aziz, C. Pixley, and K. Albin. Simplifying boolean constraint solving for random simulation-vector generation. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 23(3):412–420, November 2006.

[23] Jun Yuan, Carl Pixley, and Adnan Aziz. *Constraint-Based Verification*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[24] Zhi-Hua Zhou and Yuan Jiang. Medical diagnosis with c4.5 rule preceded by artificial neural network ensemble. *Information Technology in Biomedicine, IEEE Transactions on*, 7(1):37 –42, march 2003.