Kalpa Publications in Computing

Volume 4, 2018, Pages 248–263

28th International Workshop on
Principles of Diagnosis (DX'17)

Model-Based Diagnosis Meets Combinatorial Testing For Generating an Abductive Diagnosis Model

Ingo Pill and Franz Wotawa

Institute for Software Technology, TU Graz,
Inffeldgasse 16b/II, 8010 Graz, Austria
{ipill, wotawa}@ist.tugraz.at

Abstract

The diagnostic model is certainly a key element for any model-based diagnosis process. Experience shows though that in practice we often have no such model available for one or the other reason. The consequence for many projects is thus that we cannot draw on diagnosis processes when tackling problems. In this paper, we show how to improve available automated processes for deriving a diagnostic model from the standard simulation models that we usually create during development. We delve in particular into the question how research in the context of combinatorial testing and fault injection can help to improve the process, and consider several questions that arise.

1 Introduction

Precise problem descriptions and sound background theories for model-based diagnosis processes have been around for quite a while [1, 2, 3]. Furthermore, the scientific community has been improving on these ever since, as evident, for instance, in the work presented regularly at dedicated venues like the DX workshop. Still, the well-founded diagnosis theory and the availability of corresponding reasoning engines could not yet foster a wide-spread adoption in practice. If talking, for example, to application engineers in the Electronic Design Automation (EDA) industry, we learn that they have been adopting formal verification techniques like model-checking [4] that allow them to identify the presence of issues. When reasoning about the origins of identified issues, as of yet they usually do not use automated diagnostic reasoning though, but draw on expert knowledge in a still manual and laborious process. Support in coming up with diagnostic models would certainly be an important prerequisite for adopting also model-based diagnosis techniques for their toolset.

In [5], we surmised that in practice we often lack an appropriate model that is suitable for diagnostic purposes in a project—certainly a stumbling block for adopting model-based diagnosis (MBD). Evidently it is not an easy feat to come up with such a model, and available resources (amongst other issues like the availability of white-box knowledge for all system parts) might not enable a project manager to account for a diagnostic model. An issue here is that a diagnostic model has a different focus compared to development models created, e.g., for simulations (see Section 2). For some scenarios like formal specification development in the

context of temporal logics, we do have approaches that allow us to automatically derive models for a consistency-oriented model-based diagnosis process [6]. For other problem domains where we use, e.g, Modelica[1] or Simulink[2] simulation models for the development, we still need to improve on available work.

In this context, we proposed in [5] an automated work flow for generating an abductive diagnosis model based on fault effect descriptions. The underlying idea for extracting the fault effects was to compare simulated expected behavior with simulated faulty one that we obtained via automatically injecting faults into the correct simulation model. Isolating the individual faults' effects, we then formalized a knowledge base containing cause-and-effect rules connecting the injected faults with the isolated fault symptoms. This knowledge base can then be used in an abductive diagnosis process. An advantage of the derived abductive diagnostic model is that it is similar in its structure to the cause-and-effect rules that designers would consider during FMEA (failure mode and effect analysis) [7, 8]. With the same argument, also providing the fault models for the fault injection step should not be an issue, so that the technique can be easily integrated into existing development processes. This is backed by standards like ISO 26262[3] that suggest fault injection as a method for ensuring safety in the automotive domain.

While we showed in [5] that the work flow works out in principle, and that we can accommodate also multi-faults by design, tackling scalability in the direction of multiple faults certainly is a pressing question. That is, for n components that have m fault modes each, ideally we would have to cover m^n mode combinations for an exhaustive investigation taking fault interactions into account. Thus, an important issue left for future work was that of improving on the management regarding an efficient *and* effective exploration of the input space and the fault space. In other words, answering the question of how to deal best with the huge amount of possible combinations of faults and inputs in practice. In this paper we contribute to addressing this question, by discussing a structural and locally exhaustive approach for the exploration.

In [9], Wotawa adressed the issue of considering the input space for a familiar problem from a point of view that is relevant also for our work. That is, focusing on the scenario of testing self-adaptive systems, he proposed to draw on fault injection and combinatorial testing to come up with a test suite that is strong in identifying faults (in his scenario this resembles to testing the diagnosis engine providing the background information for the adaption process) without exploding in size and thus busting the available budget. Fault injection [10], we did consider also for [5], and pondering his findings it is evident that the reasoning behind combinatorial testing [11, 12, 13] is attractive also for our scenario. In this paper we thus elaborate on our earlier work presented in [5], and show how combinatorial reasoning can be used to conquer the simulation space in terms of faults and inputs. We furthermore identify and discuss several questions that arise with the use of combinatorial reasoning in our process.

In the remainder of this paper, we first present the basic definitions we're using in our presentation. Then, we show in Section 3 the details of our new abductive diagnosis model generation algorithm. Following an illustration of our combinatorial exploration concept for an electronic circuit in 3.1, we discuss several theoretical aspects of our approach and present another algorithm variant in Section 3.2. Finally, we provide a brief summary and outline of our future research in Section 4.

[1] http://www.modelica.org
[2] http://www.mathworks.com/products/simulink/index.html
[3] see, e.g., http://www.iso.org/iso/catalogue_detail?csnumber=43464

2 Basic Definitions

Let us start this section with defining a system's model as considered in our reasoning. Aggregating several definitions of [5], this model is defined such as to allow for simulating a system's behavior when given (a) the desired input scenario, and (b) the desired fault scenario.

Definition 1. *A system model is a tuple* $S = (COMP, MODES, \mu, \rho, I, O, M)$ *such that* $COMP$ *is a finite set of system components,* $\{ok\} \subseteq MODES$ *is the nominal (correct) mode in a finite set of modes that components can have,* μ *is a function mapping components* $c_i \in COMP$ *to their individual sets* $MODES' \subseteq MODES$*,* I *is a finite set of input signals and input variables,* O *is a finite set of observable output signals and output variables, and* M *is a simulation model that allows us to simulate the system's behavior for a finite set* $TIME$ *of discrete points in time with a simulation function* sim *as of Definition 2, taking into account also mode assignments* ρ *as of Definition 3.*

Definition 2. *Let us assume that we have a system model* $S = (COMP, MODES, \mu, \rho, I, O, M)$ *as of Def. 1, a test case* τ *defining the input values for all* $i \in I$ *over time, a mode assignment* ρ *as of Def. 3 defining the modes for all* $c_i \in COMP$ *over time, and an end time* t_e*. A simulation function* $\mathsf{sim}(S, \tau, \rho, t_e)$ *computes via* M *the values of all variables* $o \in O$ *over time (between* 0 *and* t_e*) considering (a) the test case* τ *for inputs* $i \in I$*, and (b) the mode assignment* ρ *for the components' modes.*

Definition 3. *Let* $TIME$ *be a finite set of points in time. A mode assignment* ρ *is a set of functions* $\rho_i(t)$ *which define for the* $c_i \in COMP$ *the component's mode for all time stamps* $t \in TIME$*.*

Please note that the finite set of points in time TIME will usually be determined by t_e and the simulation function's sampling frequency.

In contrast to a diagnostic model, a simulation model $M \in S$ defines a specific value for each $o \in O$ given the input and fault scenario. When reasoning with a diagnostic model on the other hand, given inputs and observed outputs, we would be interested in viable solutions (likely more than one!) for assigning the individual components' fault modes (health state variables) which in turn creates the diagnoses. As we discussed in [5], this is one of the reasons why one cannot perform diagnosis directly, e.g., in a Modelica simulator without adoptions as proposed in [14].

When we aim at testing a system, we have to exercise the system to the best of our knowledge with input stimuli that would reveal hidden issues. Sometimes it requires a certain combination of inputs and system parameters for a fault to become visible at the observable outputs. This would require us in principle to explore all such combinations when verifying the correct behavior of a system. Obviously this is infeasible, since even for a simple 64 bit adder, this would require us to cover 2^{128} bit combinations for the two 64 bit registers—when neglecting sequential effects and behavior entirely. So for n variables that can have m values each, this means m^n combinations. Combinatorial testing [11, 12] aims at addressing this issue via a structural but local approach to conquering the input and parameter space. Instead of testing all m^n combinations, there exhaustiveness focuses on local variable interactions.

In particular, this means that every x-way interaction between variables shall be considered in at least one test case. If the *combinatorial strength* x is 2, this means that between any two variables, each possible value combination for these two variables shall be featured by at least one test case in the test suite. Usually we have $x < n$ (in most cases even $x << n$), since for strength $x = n$ the concept would translate to the globally exhaustive approach.

Let us assume now that we have three Boolean (s.t. $m = 2$) input variables i_1, i_2, i_3 that can be $\bot$ (False) or $\top$ (True), and that we would like to derive a test suite with combinatorial strength 2. This means that for any variable subset of size $x = 2$ (there are three such sets: $\{i_1, i_2\}, \{i_1, i_3\}, \{i_2, i_3\}$), and for any of the $m^x = 4$ possible value combinations $(\bot, \bot|\bot, \top|\bot, \top|\top, \top)$ for these variable subsets, there shall be at least one test case featuring this exact combination. Compared to the globally exhaustive approach which would give us $m^n = 8$ test cases for three Boolean variables, the four test cases shown in the table below define a test suite with combinatorial strenth 2 for the same example. As is evident from the table, for any of the three variable subsets of size 2, the four test cases cover all four possible value combinations. Please note that we will show a larger example in Section 3.1.

	i_1	i_2	i_3
test case 1	$\bot$	$\bot$	$\bot$
test case 2	$\bot$	$\top$	$\top$
test case 3	$\top$	$\bot$	$\top$
test case 4	$\top$	$\top$	$\bot$

The idea that we exploit in this combinatorial approach is that we can cover more than one individual combination in a single test case, which allows us to reduce the number of test cases compared to the globally exhaustive approach. An immediate question is now which strength would be required in practice. There, Kuhn and colleagues showed in [15] that 6-way strength might suffice to reveal all faults, where for some domains the limit was even lower.

If we have variables with varying domains, we can use the following definition of a mixed-level covering array to describe such a "table". Please note that the term *mixed-level* indicates the support for variable sets such that each variable can have its own domain with an individual size and an individual alphabet of possible values.

Definition 4. *A mixed-level covering array* $MCA(I, (a_1, \ldots, a_k), s)$ *of strength* s *for* $k = |I|$ *variables with their individual finite alphabets* a_i *is a two-dimensional* $k \times N$ *array such that for any* $I' \subseteq I$ *such that* $|I'| = s$ *we have that every combination in the cross product of the individual alphabets of the variables in* I' *appears in at least one of the* N *rows.*

Implementing a combinatorial approach at generating test cases requires us to follow a three step process, starting with us providing an input model:

1. We have to describe the input space by listing the variables and signals that we would like to cover, and we have to specify their alphabets. For the latter, we might not want to specify the entire range of values for a variable (like all integer values), but either only the subset of values used in practice, or in the light of our discussion in [5] a finite set of exemplary values describing the variable's individual features and trajectories (like values near comparison values used in the program). This process is coined input parameter modeling in the context of combinatorial testing, and we refer the interested reader to [16] for an introduction.

2. The second step is to invoke a combinatorial design procedure like the combinatorial test generation tool ACTS [17] that derives a mixed-level covering array as of Definition 4 with the desired strength. The columns in the array relate to the considered input variables, whereas the individual rows define each some combination of value assignments to these variables considering the variables' individual alphabets. The employed combinatorial

generation procedure [17, 18, 19] ensures that each value combination required to achieve the desired strength is indeed covered by at least one row.

3. The third and final step is to consider each row as an individual test case of the test suite under construction.

In Section 3, we show how to exploit such a combinatorial approach for generating an abductive diagnosis model. That is, while we do not generate test cases like for the combinatorial testing scenario, we use the technique to generate a representative parameter versus input scenario list for our faulty behavior simulations. Each row in the obtained MCA will not specify a test case as described above, but will define the inputs and parameters for a single simulation scenario. The set of the simulation scenarios defined in the MCA will then be used to derive a diagnostic model.

Before we can show the details of our approach in Algorithm 1, let us continue our definitions with the one for the desired abductive diagnosis model. That is, in principle, our abductive diagnosis model is a knowledge base (see [20] for more corresponding definitions):

Definition 5. *A knowledge base is a tuple* $KB = (P, HYP, TH)$ *where* P *is a set of propositional variables,* $Hyp \subseteq P$ *is a set of hypotheses, and* TH *is a set of horn clause sentences over* P*.*

While a reader might now wonder why we would use propositional variables when considering simulation models that often contain a mix of continuous signals, we do this on purpose. That is, such variables state, for example, a component's finite parameters or a comparison to a constant such as to digitize the value (which is also in line with the input parameter modeling task mentioned for combinatorial testing). For a discussion of how to identify an appropriate task dependent qualitative abstraction and a corresponding automated approach, we refer the interested reader to [21].

Please note that in our context, a knowledge base's hypotheses correspond directly to the causes of an encountered issue, i.e., the diagnosis elements, or in other words the faults. Let us define now propositional horn clause abduction problems and their solutions, so that we can define diagnoses formally.

Definition 6. *Given a knowledge base* $KB = (P, HYP, TH)$ *and a set of observations* $OBS \subseteq P$*, the tuple* (P, HYP, TH, OBS) *describes a propositional horn clause abduction problem (PHCAP). A set* $\Delta \subseteq HYP$ *is a solution to a PHCAP, if and only if* $\Delta \cup TH \models OBS$ *and* $\Delta \cup TH \not\models \bot$*. A solution* Δ *is parsimonious or subset-minimal if and only if no set* $\Delta' \subset \Delta$ *is also a solution.*

A solution Δ of a PHCAP is a set of hypotheses that allows to derive the given observations via TH. Consequently, Δ is an explanation of the given observations—based on the background theory described in TH—and we refer to Δ thus also as *abductive diagnosis* or simply as diagnosis. While the definition itself does not require a diagnosis to be minimal in the first place, please note that in practice we are often interested only in subset-minimal diagnoses.

"*Determining whether a hypothesis is included in a minimal diagnosis is NP-complete*" [20]. For a comprehensive complexity analysis of logic-based abduction we refer the interested reader to [22]. If $|HYP|$ is not too high, we can compute the solutions efficiently though. Such an approach might use De Kleer's Assumption-based Truth Maintenance System (ATMS) [23, 24], encoding the observations as a single rule $o_1 \wedge \ldots \wedge o_k \rightarrow \sigma$ for $k = |OBS|$ and a newly generated proposition σ. The label of σ's node then is an abductive diagnosis for the observations, where the rules for the node labels ensure that the solution is minimal, sound, complete, and consistent. For more information we refer the interested reader to [25].

3 Generating a Model for Abductive Diagnosis Using Simulation, Fault Injection, and Combinatorial Testing Techniques

In the introduction, we outlined briefly the goal of our work. That is, like with the approach presented in [5], we aim to generate an abductive diagnosis model in a fully automated fashion. For our approach, we assume that while we have no model fit for diagnostic purposes, we do have a simulation model available that allows us to derive the system outputs for given inputs. Such a model is usually created in the system development process but has a different purpose compared to a diagnostic model. That is, it shall be as precise as possible and shall allow a designer to derive the system's reactions to any desired input scenario, rather than deriving a list of possible faults for observed faulty behavior.

Drawing on fault injection, we use this model not only to derive the correct *expected* behavior for some input scenarios, but also to simulate faulty behavior triggered by injecting faults. As outlined for Modelica models in [5], this means to define and describe alternative behavioral modes for the individual components. The set of modes for a component contains one for the nominal behavior (*ok* in our Definition 1) and some modes encoding faulty behavior like stuck-at faults for signals, empty batteries, or broken connectors. Such data is usually considered also in FMEA, so that it is available in practice. Otherwise, a designer has to provide the desired fault modes. Using a special variable for each component to switch between its modes, we can define any possible fault scenario (as allowed by the specified modes) via specific assignments to these mode variables. Providing the same input stimulus to a correct and a faulty system model, we can check for deviations (see [5] for a discussion of corresponding techniques), and if we detect such a deviation, we extract a rule expressing that the activated faults lead to the identified deviations.

The concept for automatically deriving an abductive diagnosis model is now to activate different fault scenarios, and to extract the corresponding fault and effect rules, aggregating them in a knowledge base. The algorithm for Modelica models that we presented in [5] limits the fault scenarios to single fault activations. While this has the advantage that it requires only a limited number of simulations per input scenario ($(m-1)*n+1$ for n components with m modes each), this also means that fault interactions would not be considered in the knowledge base. Compared to the algorithm in [5], we thus aim to consider fault interactions in the presented work. Conducting m^n simulations per input scenario so that we would consider all fault interactions is, however, certainly unattractive—simply due to the sheer amount of required resources.

Inspired by the concept of combinatorial testing and Wotawa [25] using the concept for testing a diagnosis engine (i.e. the system self-adaption) in a setting as outlined in [26], we do aim to exploit combinatorial testing in order to keep the amount of required simulations in manageable regions. The key question now is what we would consider as the "input space" of these simulations, such as to consider in a combinatorial exploration. In principle, we have three groups of "inputs" for a simulation:

1. **system inputs:** A system input like the register value of a microprocessor, the analog measured temperature for a thermostat, or some input of a Boolean circuit represents an interface via which the considered system recognizes its environment. Depending on the scenario and the model, this usually means some temporal behavior, either in continuous analog form or a clocked digital signal. For scenarios like a Boolean circuit, there would not be such a temporal aspect though when we consider it at the digital level.

2. **system parameters**: Components might have parameters in order to configure the component instances in a model. For example, resistors having a nominal value of 100Ω and a tolerance of 5% will have an actual value in the range (95...105)Ω. System parameters will allow a designer to explore some options for their simulations, for example via considering the set $\{95, 100, 105\}$ as alphabet (see Def. 4) for the resistor value system parameter. Usually we would assume such a parameter to stay constant during a simulation.

3. **mode assignment variables:** These variables allow us to activate the individual component faults. In principle we could have static faults, but also intermittent ones.

The first group describes in principle the system's functional input model, i.e., those inputs that the system was designed to act upon. The second and third group though could be seen as exogenous influence on the system that we did not add by the functional design, but which were added to the model for simulating reality and exogenous events not under control. In terms of our system model as of Definition 1, the first two groups would form I, while the third group is covered by the mode assignments (see also Definition 3).

Before we present our new algorithm, let us make the following assumptions. In order to support static or intermittent faults, a user can specify a fault activation pattern which is a function $f(t)$ that maps any $t \in \mathit{TIME}$ to $\top$ (True) if the fault shall be active for time stamp t, or $\bot$ (False) if it shall be inactive for t. For each input $i \in I$, we specify an alphabet $\Sigma(i)$ of input sequences such that each such sequence $\sigma \in \Sigma$ defines input i's value for each $t \in \mathit{TIME}$ and us denoting the individual values with $\sigma(t)$. Aside a simulation function as of Definition 2, we also require there to be a function $\mathsf{diff}(\mathbf{B}_{corr}, \mathbf{B}_{faulty})$ for comparing simulation results and identifying deviations. As discussed in detail in [5], this comparison function deserves special attention and there are two basic implementation concepts.

Algorithm 1 encodes the various steps of our generation process. In line 1, we initialize the sets forming the desired knowledge base. Lines 2 and 3 concern the definition of the variables and their alphabets as considered when creating the MCA in Line 4. Each individual row in the MCA C defines all the needed inputs and parameters for one simulation run. Consequently, V contains all input variables and also the mode assignment variables h_i for the individual components in COMP. For all inputs $i \in I$ we assume that there is an alphabet of input sequences to choose from (so that we choose one behavior out if this alphabet as input for i in a simulation). The alphabets for the mode assignment variables are determined by $\mu(c_i)$, which is a function of S that maps the individual components to their individual mode subsets (see Def. 1). We furthermore assume that we can represent any sequence in an input's alphabet by propositional variables (which we done in Line 17).

In Line 5, we store in R the number of MCA rows—to be used as boundary for the number of iterations of the Algorithm's core loop described in Lines 7 to 28. The function defined in Line 6 deserves special attention and is used to extract the sequence or mode for some $i \in I$ respectively $c_i \in \mathit{COMP}$ as given in a single row of the MCA C. The core of Algorithm 1 is defined in Lines 7 to 28. There we consider the individual rows of C iteratively, and extract ρ' as the temporal mode assignment (in Lines 9 to 12), Δ as the corresponding set of active fault modes (all assignments other than the nominal one)—also in Lines 9 to 12—, and τ as the input scenario (Lines 14 to 16)—all from the data given in the currently considered row as selected in Line 7 with variable r. Every proposition required for describing the extracted input scenario is then added to P in line 18. In Lines 19 and 20 we simulate the correct ($\mathbf{B}_{corr}$) and faulty ($\mathbf{B}_{faulty}$) behaviors, comparing them afterwards in Line 21, isolating a possibly empty set of deviations D. Please note that ρ for defining a constant assignment of mode ok to all component's mode variables for simulating $\mathbf{B}_{corr}$ is provided via the algorithm's input

Algorithm 1 Rule extraction when simulating the combinatorically explored input space

Require: **(1)** A model $(COMP, MODES, \mu, \rho, I, O, M)$ as of Def. 1 where ρ is defined such that each component is in mode ok all the time, **(2)** a fault activation pattern f, **(3)** a list A such that $\Sigma_i = a_i \in A$ is input $i \in I$'s alphabet,**(4)** a function $MCA(V, A', s)$ that generates a mixed-level covering array (see Def. 4) of strength s for the set of variables V with $a_i \in A'$ being $v_i \in V$'s alphabet, **(5)** the desired strength s, **(6)** a simulation function $\mathsf{sim}(\mathbf{P}, \tau, \rho, t_e)$ and end time t_e, and **(7)** a function $\mathsf{diff}(\mathbf{B}_{corr}, \mathbf{B}_{faulty})$ comparing simulation results and identifying deviations.

Ensure: A KB (P, HYP, TH)

1: Let P, HYP, and TH be empty sets.
2: $V \leftarrow I \cup \{h_i | c_i \in COMP\}$ % h_i is c_i's mode variable
3: $A' \leftarrow A \cup \{\mu(c_i) | c_i \in COMP\}$
4: $C = MCA(V, A', s)$
5: $R \leftarrow$ number of rows in C
6: Let $val(v \in V, r)$ be a function returning the value of v in row r of C
7: **for** r in range(1, R) **do**
8: Let ρ', Δ, and τ be empty sets
9: **for** c_i in $COMP$ **do**
10: **if** $val(h_i, r) == ok$ **then** $\rho'_i = ok$ $(\forall t \in TIME)$
11: **else** Let $\Delta \leftarrow \Delta \cup \{h_i\}$, and let $\rho_i = f'$ be derived from f such that $\top$ is replaced with $val(h_i, r)$ and $\bot$ with ok
12: **end if**
13: **end for**
14: **for** i in I **do**
15: $\tau \leftarrow \tau \cup val(i, r)$ % add the sequence for i
16: **end for**
17: Let τ_p be the propositional representation of τ.
18: $P \leftarrow P \cup \tau_p$
19: Let $\mathsf{sim}(\mathbf{P}, \tau, \rho, t_e)$ be $\mathbf{B}_{corr}$.
20: Let $\mathsf{sim}(\mathbf{P}, \tau, \rho', t_e)$ be $\mathbf{B}_{faulty}$.
21: Let D be the result of $\mathsf{diff}(\mathbf{B}_{corr}, \mathbf{B}_{faulty})$ considering variables in O only.
22: Add all elements of D to P.
23: Let Δ_P be the *and* $(\wedge)$ of all h_i in Δ
24: Add the propositions in Δ_P to P and HYP.
25: **for** d in D **do**
26: Add the rule $\Delta_P \wedge (\bigwedge_{p \in \tau_p} p) \rightarrow d$ to TH
27: **end for**
28: **end for**
29: **return** (P, HYP, TH)

parameter S. The deviations described in D represent the symptoms caused by the fault. For being able to add our corresponding cause and effect rules to TH in Lines 25 to 27, we have to add propositions needed to encode $d \in D$ to P, which we do in Line 22.

Finally, the algorithm returns a knowledge base (P, HYP, TH) in Line 29. We can use this knowledge base to create PHCAPs as of Definition 6 for some observations OBS. The solutions to these PHCAPs then provide abductive diagnoses explaining the observations, based on the knowledge about fault effects obtained by Algorithm 1.

It is easy to see that our algorithm works as expected by construction. If we now assume that sim and diff terminate, we have that Algorithm 1 algorithm terminates, since the contained loops iterate over finite sets only. For considering the algorithm's complexity, let us assume unit time for sim and diff, so that the complexity is entirely determined by the number of rows in the MCA C (lines 9 to 16 for extracting the data from the row run linear in the amount of C's columns). While the simulation time will amount for most of the experienced run time in practice, these simulations are required to be conducted only once—when establishing the abductive diagnosis model—and not at run time when performing diagnosis.

If we consider the concept of Algorithm 1 in detail, some question arising immediately is that of whether we should really cope with the three input types identified above in the same manner. In particular, if considering this question a bit, we can unveil several arguments in favor of treating them differently. Before pondering this question closely in Section 3.2, let us illustrate the combinatorial concept of Algorithm 1 for the example of an analog circuit in Section 3.1.

3.1 Deriving the set of simulations as of Algorithm 1 for an RC circuit

Let us consider the example of a small electronic circuit as showcase for the determination of the simulation parameters, i.e., the MCA C, with Algorithm 1. In Figure 1, we show the diagram of an example RC circuit comprising four components: a battery B, a changeover switch S, a resistor R, and a capacitor C. Please note that in this section C thus refers to this capacitor—rather than the MCA—,that is, if not specified otherwise.

As four second input sequence for the switch S, we assume that it is open initially, gets closed after one second (1s), is opened again after another second (timestamp 2s), and stays open until the end of the scenario. Thus, while the capacitor is empty for the first second, it gets loaded when we close the switch. When opening the switch again at timestamp 2s, the capacitor is almost full, but now the resistor R starts to discharge C. In Figure 2, we can see the simulated resulting ordinary circuit behavior for this input scenario, i.e., the voltage drop v_C and the position of the switch S over time. For this simulation, we assumed a battery voltage v_B of $5V$, a resistor value r of $10k\Omega$, and a capacitance c of $10\mu F$. In reality, the circuit's behavior will obviously depend on its current health state and the component parameters. The latter might vary due to manufacturing tolerances or deterioration. In the following, we assume that the battery, the resistor, or the capacity might break, and that there is some variation in the components' parameters. For simplicity, we assume though that the switch S never fails.

Since we have a single input sequence for the switching signal, we do not need to consider it as input in our combinatorial input model. For the component parameters r, c, and v_B, we con-

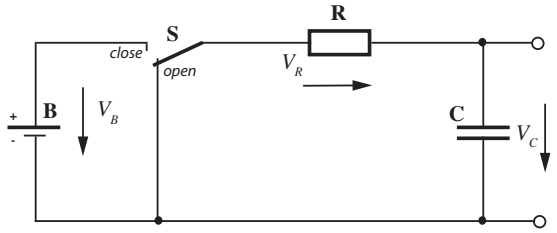


Figure 1: A circuit with a battery B, a changeover switch S, a resistor R, and a capacitor C.

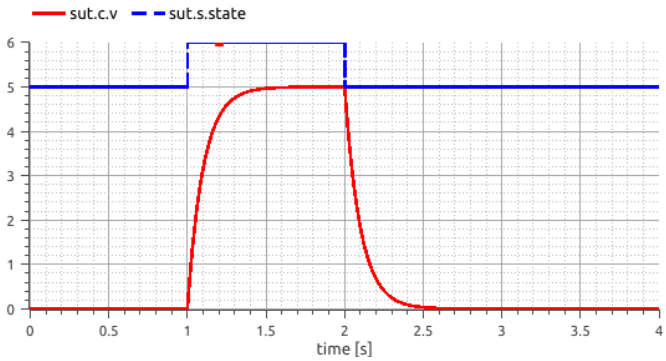


Figure 2: The ordinary behavior of the simple electric circuit from Figure 1.

sider for their individual alphabets the respective nominal values and a 10% tolerance in both directions (resulting in the set $\{4.5V, 5V, 5.5V\}$ for v_B). Regarding the mode assignment, we assume that the battery can be full (*ok*) delivering its nominal voltage, or *empty* such that v_B is 0. A resistor or capacitor can have three modes: it is *ok* when it works as expected, or it can be *short* or *broken*. For the former fault case, we would not see any voltage drop at R, whereas for the latter, we would not have any current flowing through the resistor. As mentioned previously, there are no fault modes except *ok* for the switch S, so that we do not include S's mode in the input model. Summing up, we get the input model for this scenario as outlined in Table 1.

Considering the sizes of the individual variables' alphabets, a globally exhaustive approach would have to cover $3 \times 3 \times 3 \times 2 \times 3 \times 3 = 486$ combinations. If we would restrict ourselves to 2-way component interactions, the ACTS tool [13] would compute 15 different scenarios for this input model and strength 2, resulting in a significant reduction in the number of considered scenarios—15 vs. 486, or in other words a reduction by 96.9%. In Table 2 we report all these scenarios, where we have several ones for which it is assumed that two components behave correctly. In Figures 2 and 3 we can see the simulated nominal behavior and the faulty behaviors for scenarios 3 and 10 (as described in Lines 3 and 10 of Table 2—labeled 3 and 10 in column#) respectively. There we can see that we obtain different curves that we can distinguish with the techniques outlined in [5]. Only for scenarios 3,4, and 6 we would not be able to distinguish the simulation results. Please note that we assumed a constant fault activation pattern for these simulations.

Table 1: Modeling the input space for the RC circuit.

category	variable	alphabet
inputs	–	–
parameters	v_B	$4.5V$, $5V$, $5.5V$
	r	$9k\Omega$, $10k\Omega$, $11k\Omega$
	c	$9\mu F$, $10\mu F$, $11\mu F$
mode assignments	B	*ok*, *empty*
	R	*ok*, *short*, *broken*
	C	*ok*, *short*, *broken*

Table 2: Scenarios for the RC circuit and strength 2.

#	v_B	r	c	B	R	C
1	4.5V	9kΩ	10 μF	*empty*	*short*	*short*
2	4.5V	10kΩ	11 μF	*ok*	*broken*	*broken*
3	4.5V	11kΩ	9 μF	*empty*	*ok*	*ok*
4	5V	9kΩ	11 μF	*ok*	*ok*	*short*
5	5V	10kΩ	9 μF	*empty*	*short*	*broken*
6	5V	11kΩ	10 μF	*ok*	*broken*	*ok*
7	5.5V	9kΩ	9 μF	*empty*	*broken*	*short*
8	5.5V	10kΩ	10 μF	*ok*	*ok*	*broken*
9	5.5V	11kΩ	11 μF	*empty*	*short*	*ok*
10	4.5V	9kΩ	9 μF	*ok*	*short*	*ok*
11	5V	10kΩ	10 μF	*empty*	*broken*	*ok*
12	5.5V	10kΩ	11 μF	*empty*	*ok*	*short*
13	5.5V	11kΩ	11 μF	*ok*	*broken*	*short*
14	5V	9kΩ	11 μF	*empty*	*broken*	*broken*
15	4.5V	11kΩ	11 μF	*ok*	*broken*	*broken*

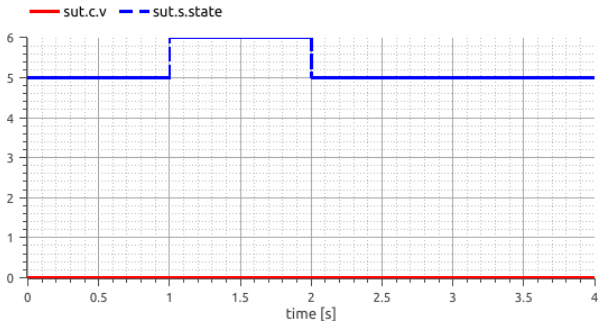
3.2 Analyzing the concept and focusing the scope of the combinatorial exploration in an algorithmic variant

For exploring the arguments about treating the *system input*, *system parameter*, and *mode assignment* variable groups differently, let consider some aspects of these three groups individually, starting with those for the third one.

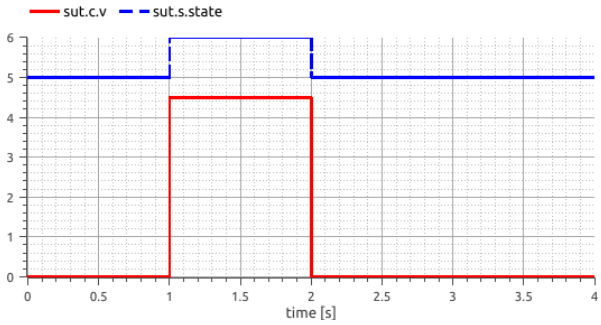
The *mode assignment variables* obviously correspond to the health state variables considered in the diagnosis theory [2, 3] when using a strong fault model (i.e., when defining abnormal behavior as opposed to the standard normal/abnormal predicates). In model-based diagnosis, besides focusing on subset-minimal solutions, we also often limit the cardinality of computed diagnoses. The rationale behind this is that we consider it unlikely that a lot of components fail at the same time, and thus limit the search to, e.g., triple fault diagnoses. This allows us to strategically limit the search space that is otherwise exponential in the amount of health state variables. In combinatorial testing, we have a similar limit, but which focuses on the interactions of component values. There the idea then is to have one test case combine several of those interactions on different variable subsets. If we now do this for the mode assignment variables (and thus in principle to diagnoses), we would have to investigate two questions:

1. *Is the simulation space really good enough to trigger the possible deviations?* In principle this seems to be the case, since the argument would be similar to the one why the approach would be good for testing/triggering hidden faults in testing. While fault masking could avoid triggering effects in some cases, in line with the argumentation regarding combinatorial testing, we would expect the *set of* simulations to cover (in other words produce) the available set of effects.

   This could directly help in configuring the fault detection steps, i.e., when defining the allowed tolerances for some signals in order to detect the presence of a fault. Only when these would be violated, we would trigger a diagnostic process. Aside for tuning tolerance values, these data help when defining a corresponding temporal band sequence [27, 28] for



(a) scenario three



(b) scenario ten

Figure 3: Simulation results for two RC circuit scenarios.

fault detection, i.e., when approximating upper and lower hulls for the expected corridors for the correct individual signals based on the simulation results with expected parameter variations.

2. *How to generate the rules for a knowledge-base that we can use for abductive diagnosis?* For single fault activation, rules of the type "fault X being active implies effect Y for input scenario Z" are attractive. If we now follow a combinatorial approach for the mode assignment variables, this translates to stating "the union of all active faults together with the given inputs implies the recognized effects" (see line 26 in Algorithm 1). Now, i.e., in Algorithm 1, the health state variables are not "activated" in isolation though. In terms of the diagnostic reasoning allowed by the derived knowledge base, this means that we could deduce only diagnoses that are the union of some of the mode assignments in the MCA C's rows. Depending on the whole set of variables V and their domains A' as fed into the mixed-level coverage array algorithm (these would directly influence the total number of rows, which in turn has an impact on the mode assignments of the individual rows), it is less clear what the effects would be on the diagnostic reasoning. It is important to note at this point that strength x only defines that all x-way interaction of variable subsets of size x appear in some row, but not how independent ones (for different or intersecting subsets) are combined to enhance efficiency, i.e., to get fewer rows.

The second group of variables, i.e., the *system parameters* (part of I for both algorithms), are certainly a group of variables that is in line with the variables that combinatorial testing was

designed for. Of course we have to take care when designing the alphabet as of Definition 4 for these variables. However, available work from Sachenbacher and Struss can help in finding an appropriate abstraction in an automated fashion [21]. Please note that in most cases, we could certainly define these variables' alphabets as sets of elements rather than sets of sequences.

The first variable group of *system inputs* is also prone to be tackled with a combinatorial exploration. Considering a Boolean circuit, we could even consider the full input domains of these variables. When it comes to continuous signals and temporal behavior, obviously the input space for these variables grows significantly. The before-mentioned input modeling task is one where we should definitely take care thus, possibly drawing on work on automated abstraction like [21]. For defining our algorithms, we took an approach where we could define several individual patterns for each variable, to be chosen from in the generation process. This implements a trade-off between conquering all possible values and having a presentation that allows us to employ combinatorial exploration. Some further extension in this respect could be to allow patterns over multiple variables. That is, obviously there are several ways to handle a combinatorial exploration for these variables, and our algorithms can be easily adapted to accommodate other strategies.

Summarizing our thoughts we see indeed some reasons for treating the variable groups differently. Implicitly we do that already with the inputs in that we "mix" sequences for the system inputs and constants for the system parameters. Considering our discussion of the two questions regarding the mode assignment variables, some option where we could add constraints on the mode assignments for some simulation (i.e., those in a single row in the MCA C) could improve our confidence in the diagnostic model. That some MCA generation tools like ACTS allow us to specify constraints for the MCA generation is thus certainly welcome.

For a simple solution, we could express for instance, a maximum limit of concurrently faulty components in order to avoid uncertainty or a lack of confidence in this respect. Please note that we could also think of more elaborate solutions in this respect. For example, if we derive dependency graphs for the individual outputs and we see that some components can be faulty concurrently without affecting the individually affected output signals, we could support a more elaborate solution where we define such kernels that can be assigned independently. If now our rule extraction as of Line 26 in Algorithm 1 would take care of these kernel descriptions, we could overlay assignments in these kernels in a single simulation without loosing preciseness for the cause-and-effect rules added to the knowledge base.

But for now, let us define a simple variant of Algorithm 1, such that we can set a limit on the maximum number of components that simultaneously feature some mode $\neq ok$.

Definition 7. *Let Algorithm 2 be like Algorithm 1, but using a different MCA generation function* $MCA(V, A', s, CONS)$ *that deviates from Definition 4 by having an additional parameter CONS. This parameter allows us to specify a set of constraints for the generation process. For Algorithm 2, this set contains only constraints that limit the cardinality of the set* Δ *(as considered in line 23 in Alg. 1) containing all components whose mode is not ok to some value* s'*. This value* s' *shall be determined by a new parameter of Algorithm 2, and shall be referred to as* mode assignment strength*.*

Considering the changes for Algorithm 2 (which concern adding and handling the mode assignment strength s' only), we see that they do not change the termination or correctness aspects of the algorithm. Only the number of simulations as specified in the MCA will vary.

For the example considered in Section 3.1, we would get 15 simulation scenarios with the combinatorial exploration concept outlined for Algorithm 2 and a mode assignment strength of 1. For implementing the mode assignment strength, we added the constraint ($B = ok \wedge R =$

Table 3: The different scenarios computed for the RC circuit when desiring combinatorial strength 2 and adding a constraint that requires two components to work as expected.

#	v_B	r	c	B	R	C
1	4.5V	9kΩ	$10\mu F$	*ok*	*short*	*ok*
2	4.5V	10kΩ	$11\mu F$	*ok*	*broken*	*ok*
3	4.5V	11kΩ	$9\mu F$	*ok*	*ok*	*short*
4	5V	9kΩ	$11\mu F$	*ok*	*ok*	*broken*
5	5V	10kΩ	$9\mu F$	*ok*	*short*	*ok*
6	5V	11kΩ	$10\mu F$	*ok*	*broken*	*ok*
7	5.5V	9kΩ	$9\mu F$	*ok*	*broken*	*ok*
8	5.5V	10kΩ	$10\mu F$	*ok*	*ok*	*short*
9	5.5V	11kΩ	$11\mu F$	*ok*	*short*	*ok*
10	4.5V	9kΩ	$9\mu F$	*empty*	*ok*	*ok*
11	5V	9kΩ	$11\mu F$	*ok*	*ok*	*short*
12	4.5V	10kΩ	$09\mu F$	*ok*	*ok*	*broken*
13	5.5V	11kΩ	$10\mu F$	*ok*	*ok*	*broken*
14	5V	10kΩ	$10\mu F$	*empty*	*ok*	*ok*
15	5.5V	11kΩ	$11\mu F$	*empty*	*ok*	*ok*

$ok) \vee (B = ok \wedge C = ok) \vee (R = ok \wedge C = ok)$ when invoking the ACTS tool. While the number of scenarios is thus equal to the one we got without the constraint, we can see from Table 3 that we indeed have *only* scenarios where no more than one component is faulty. If we consider the algorithms, this could have an impact on the diagnostic granularity of the resulting knowledge base.

Comparing the results of the two algorithms would certainly give us further insights into the effects on the diagnostic capabilities of the derived abductive diagnosis models as resulting from the simulation efficiency achieved in their generation. Such experiments have to be chosen wisely though, as we discuss in our summary in the next section.

4 Summary

In this paper we extended previous work on automatically generating abductive diagnosis models from available simulation models. In order to strengthen the abductive diagnosis process by considering fault interactions, we employ a combinatorial exploration of the huge amount of possible simulation configurations. We showed how to draw on these techniques in order to avoid busting available project resources, with the advantage being that this strategic and locally exhaustive idea can be fully automated.

We furthermore discussed several aspects of adopting such an exploration in our algorithm and unveiled several interesting questions where corresponding answers from future research will allow to confirm or refine our surmises, and will also provide further potential for optimizing the diagnostic model generation process.

While our presentation shows that the concept works in principle, an extensive set of examples will be needed to provide showcases and to understand which strengths or algorithmic variants as discussed in the paper should be preferred. As Kuhn showed for combinatorial testing in [15], this depends on the domains, so that we will have to explore several domains

when evaluating the effects of problem sizes, observation sizes, abstraction levels, combinatorial strength, mode assignment strength, options for implementing the function diff and other parameters like repeatability of the MCA algorithm deployed. Such experiments will also unveil pitfalls in terms of input-space models, and whether specific variable groups as mentioned in the paper should be considered in specific manners.

The key result of the experiments will be the understanding which strength k is needed to capture enough behavioral details with our simulations, so that in the end we come up with an abductive diagnosis model that shows the desired effectiveness. Consequently, these results will enable us to gain more confidence in understanding the achieved compromise in respect of what the diagnostic model can deliver, and what we might miss due to the obbiously incomplete but systematic exploration of a system's behavior.

Acknowledgments

Parts of this work were created in the ENABLE-S3 project that has been receiving funding from the ECSEL Joint Undertaking under grant agreement No 692455. This joint undertaking receives support from the European Union's HORIZON 2020 research and innovation program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, and Norway. ENABLE-S3 is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and April 2019. More information is available at https://iktderzukunft.at/en/.

References

[1] Randall Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[2] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[3] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[4] Edmund M. Clarke, Masahiro Fujita, Sreeranga P. Rajan, Thomas W. Reps, Subash Shankar, and Tim Teitelbaum. Program slicing of hardware description languages. In *Conference on Correct Hardware Design and Verification Methods*, pages 298–312, 1999.

[5] B. Peischl, I. Pill, and F. Wotawa. Abductive diagnosis based on modelica models. In *27th Int. Workshop on Principles of Diagnosis (DX)*, 2016. No archival proceedings.

[6] I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *International Joint Conference on Artificial Intelligence*, pages 1053–1059, 2013.

[7] P. G. Hawkins and D. J. Woollons. Failure modes and effects analysis of complex engineering systems using functional models. *Artificial Intelligence in Engineering*, 12:375–397, 1998.

[8] M. Catelani, L. Ciani, and V. Luongo. The FMEDA approach to improve the safety assessment according to the IEC61508. *Microelectronics Reliability*, 50:1230–1235, 2010.

[9] F. Wotawa. Testing self-adaptive systems using fault injection and combinatorial testing. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 305–310, 2016.

[10] J. Voas and G. McGraw. Software fault injection: inoculating programs against errors. *Software Testing, Verification and Reliability*, 9(1):75–76, 1999.

[11] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Trans. Softw. Eng.*, 23(7):437–444, July 1997.

[12] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. Sp 800-142. practical combinatorial testing. Technical report, 2010. available via http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf.

[13] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn. Combinatorial testing of ACTS: A case study. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 591–600, April 2012.

[14] K. Lunde. Object oriented modeling in model based diagnosis. In *Modelica Workshop 2000 Proceedings*, pages 111–118, 2000.

[15] R. Kuhn, R. Kacker, Y. Lei, and J. Hunter. Combinatorial software testing. *Computer*, 42(8):94–96, 2009.

[16] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. *Introduction to Combinatorial Testing*. Chapman & Hall/CRC, 1st edition, 2013.

[17] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn. Acts: A combinatorial test generation tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375, March 2013.

[18] Victor Kuliamin and Alexander Petukhov. *Covering Arrays Generation Methods Survey*, pages 382–396. 2010.

[19] C. Nie, J. Jiang, H. Wu, H. Leung, and C. J. Colbourn. Empirically identifying the best greedy algorithm for covering array generation. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 239–248, March 2013.

[20] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Hypothesis classification, abductive diagnosis and therapy. In *First International Workshop on Principles of Diagnosis*, Menlo Park, July 1990. Also appeared in Proceedings of the International Workshop on Expert Systems in Engineering, Lecture Notes in Artificial Intelligence, Vol. 462, Vienna, September 1990, Springer-Verlag.

[21] Martin Sachenbacher and Peter Struss. Automated qualitative domain abstraction. In *International Joint Conference on Artificial Intelligence*, pages 382–387, 2003.

[22] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, January 1995.

[23] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[24] Johan de Kleer. A general labeling algorithm for assumption-based truth maintenance. In *Proceedings AAAI*, pages 188–192, 1988.

[25] Franz Wotawa. Failure mode and effect analysis for abductive diagnosis. In *Proc. Intl. Workshop on Defeasible and Ampliative Reasoning (DARe-14)*, 2014.

[26] Mihai Nica, Ingo Pill, and Franz Wotawa. Testing diagnostics components supervising functional safety requirements. In *Annual Conference of the Prognostics and Health Management Society (PHM)*, 2015.

[27] Etienne Loiez and Patrick Taillibert. Polynomial temporal band sequences for analog diagnosis. In *15th International Joint Conference on Artificial Intelligence, IJCAI 97*, pages 474–479, 1997.

[28] Rim Mrani Alaoui, B. O. Bouamama, and P. Taillibert. Diagnosis based on temporal band sequences - a empirical comparison to statistical approachs. In *Proceedings of the Automation Congress*, volume 17, pages 435–440, June 2004.