On Using an I/O Model for Creating an Abductive Diagnosis Model via Combinatorial Exploration, Fault Injection, and Simulation

Ingo Pill[1] and **Franz Wotawa**[1]
[1]Institute for Software Technology, TU Graz
Inffeldgasse 16b/2.Stock, 8010 Graz, Austria
{ipill, wotawa}@ist.tugraz.at

Abstract

In practice, we often lack a detailed diagnostic model and also the data (or resources) to create it. Obviously, this is quite a hurdle for deploying automated diagnostic reasoning. In order to overcome it, we proposed in recent work to employ a combinatorial behavior exploration concept for automatically generating an abductive diagnosis model. In the proposed approach, we basically compare correct and faulty behavior that we derive by drawing on fault injection and simulation techniques. We then aggregate data about which specific sets of faults would lead to these or those deviations in the behavior, and finally encode them in an abductive diagnosis model. Since the behavioral space as resulting from the individual domains for the various inputs, system parameters, and injected faults tends to be rather huge, we proposed first concepts to explore it combinatorially. In this manuscript, we delve deeper into the question of how to efficiently explore sequential system behavior in such an approach. That is, while we initially assumed that a user creates a finite alphabet of representative sequences to be covered, in this paper we investigate the use of an abstract input or I/O model to derive such an alphabet, and discuss resulting opportunities and ramifications for the concept.

1 Introduction

Model-based diagnosis [1; 2; 3] is a very powerful and well structured approach to isolating explanations for encountered problems. The derived diagnoses tell us which components (or faults in general) could be responsible in their union for the encountered issue. Via the required diagnostic model we take the known system structure and expected behavior into consideration, exhaustively explore the entire behavioral space described in it, and come up with a complete set of explanations considering the explanations' ramifications for the entire behavior. Despite the basic processes and techniques being available for quite some time, adoption in practice is often limited, since we might not have the detailed data or resources available that we would need for creating the required diagnostic model. The first could be related to closed third party components, the latter to cost pressure or the lack of the appropriate personnel. Statistical approaches like SFL [4; 5] offer solutions to this problem in that they consider execution data from failing and correct behavior and, based on their involvement in the individual execution, aim to rank system components according to their suspiciousness of being responsible for the observed issue(s). The underlying reasoning concept is based on abstract data about which components have been involved in the individual executions though, rather than on a detailed model. Extensions like SENDYS [6] augment the statistical reasoning with structural information in order to improve preciseness by drawing on readily available structural information like slices [7].

As a prospective alternative, we have been discussing the option of automatically constructing an abductive diagnosis model via simulation and fault injection [8; 9]. Such an abductive model would contain rules similar to FMEA [10] data that engineers are familiar with, in that it aggregates cause-and-effect rules describing under which assumptions a fault would lead to which symptoms (in other words "deviations") in the behavior. From encountered symptoms and given system parameters, we can then reason backwards via these rules in order to identify the desired set of explanations. The underlying idea of the proposed concept was to simulate correct as well as faulty behavior, where we would trigger faults via fault injection techniques. In principle, isolating the deviations would allow us to come up with a list of rules of the type *fault x under assumptions Y would lead to symptoms Z (with Y and Z being sets)*, where these rules then represent the basis for the abductive diagnosis model [8].

The rather simple and intuitive concept has some intricacies though, which we need to address for being able to effectively deploy the concept in practice. So it is apparent that the behavioral space to be covered by the simulations would be infinite even for a simple example where a single system input variable has a continuous domain. And there is, e.g., also the question of how to concretely implement the comparison of behavior. Like which margin we would allow when considering variables in a continuous domain—both in terms of the absolute signal values themselves, and also regarding a time delay. A first basic concept was proposed in [8], and in [9] we extended the approach by suggesting to use a combinatorial exploration concept that allowed us to consider also sets of faults instead of single faults only. That is, while it is easily seen that exhaustively covering all combinations of system parameters, input signal scenarios and fault combinations is infeasible in practice, the local exhaustiveness of a combinatorial exploration concept would support us in coming up with a necessarily incomplete, but structural approach at ex-

ploring the combinations. In particular this means that for some combinatorial strength s, every combination of values for every variable subset of size s serves as (partial) input for at least one simulation. The aim then is to overlay these partial variable assignments in such a way that we would need as few simulations as possible. As proposed in [9], we can derive a corresponding set of simulation configurations via the generation of mixed-level covering arrays (MCAs) as used also in combinatorial testing [11; 12]. Continuous variable domains have to be abstracted using, e.g., techniques as proposed in [13]. While we will rehearse the overall concept briefly in the next section, we would like to refer the interested reader to [9] for deeper discussions of several questions in this respect—like how to treat the variable space and the resulting ramifications.

A proposal we made in [9] is that the individual variable groups (like health state variables or input variables) should be treated individually when creating the MCA. This would allow us a more fine grained control of the activation of faults, for instance, since we might want to consider the *fault strength* in isolation from the combinatorial strength when creating an MCA. Another suggestion was to establish finite alphabets also for sequential behavior such that some input sequence for an input signal would be represented by a single letter in the alphabet for this input. This allowed us to show that, in principle, we can use the combinatorial exploration concept for creating an abductive diagnosis model.

In our discussions in [9], we mused that such limitations might not be ideal though, and there would still be the open question of how to handle the compatibility of the various sequences chosen for the individual input signals. One solution where we would tackle the latter is to derive an alphabet of input scenarios covering more (or all) input signals, which is obviously not a trivial task by itself. In this paper, we investigate exactly this task and related issues. That is, we propose to use an input model, either derived from an I/O representation of the desired system behavior, or defined directly. In particular, we investigate how we could create and use such a model in order to support creating the data for the MCA generation step. At least the data for the input model would be needed also when designing the system and deciding which components shall interact in which way, so that they should be available in a more or less formal form.

2 The Problem and Preliminaries

Let us briefly rehearse our basic definitions and the combinatorial exploration concept as proposed in [9]. Aggregating several definitions of [8], we define a system model such that it allows us to *simulate* a system's behavior when given (a) the desired input scenario, *and* (b) the desired fault scenario. This means that compared to a model describing only the nominal behavior, we need the system model to consider a set *MODES* of fault modes that the individual components can feature, and which we can activate individually for the various system components. Via the concepts of mutations [14] and fault injection [15], we can create such models easily, e.g., for Simulink[1] models with tools like SIMULTATE [16]. Please note that we assume discrete time, where the finite set TIME of time steps is usually determined by the simulation scope ranging from $t = 0$ to end time t_e, and the simulator's sampling frequency.

[1] https://www.mathworks.com/products/simulink.html

Definition 1. *[9] 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. *[9] Let us assume that we have a system model* $S = (COMP, MODES, \mu, \rho, I, O, M)$ *as of Definition 1, a test case* τ *defining the input values for all* $i \in I$ *over time, a mode assignment* ρ *as of Definition 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. *[9] 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.*

A simulation model $M \in S$ allows us to derive a system's output (values for all $o \in O$ for all $t \in$ TIME) if we provide an input scenario and in our case also a fault scenario. That is, if and only if all input signals and (fault) variables, i.e., everything aside the outputs, are given. When reasoning with a diagnostic model on the other hand, inputs and observed outputs are the basis to reason about viable solutions in terms of the individual components' fault modes (health state variables, or AB predicates in [2]). These diagnoses then shall offer explanations about which components could have behaved erroneously in order to explain the observed, faulty output behavior. As we discussed in [8], this difference in the models is one of the reasons why usually one cannot perform diagnosis in a simulator directly, that is, without adoptions like proposed in [17] for Modelica[2].

In [8], we proposed an automated workflow that would allow us to automatically create an abductive diagnosis model from a system model as described in Def. 1. The underlying idea is to take given input scenarios and after injecting single faults we would isolate the individual faults' effects on the simulated behavior, in order to come up with cause-and-effect rules relating faults and symptoms. Aggregating the rules in an abductive diagnosis model then allows us to automatically reason about diagnoses. In principle, we formulate such an abductive diagnosis model as a knowledge base (see [18] for more knowledge-base definitions):

Definition 4. *[9] 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*.*

A reader might now wonder why we would use propositional variables when reasoning with simulation models that most likely feature continuous signals. The reason is that even for a single continuous variable, the simulation space would be infinite if we'd like to exhaustively cover all possible values. In practice the actual variable values are often

[2] https://www.modelica.org/

limited though, and the propositions are used to describe a signal's finite set of possible values, or a comparison to constants such as to digitize the value via a qualitative abstraction. For a discussion of how to identify an appropriate task dependent qualitative abstraction and a corresponding automated approach, we refer the interested reader to [13].

In our case, the hypotheses in the knowledge base correspond directly to the possible causes of an encountered issue. That is, the elements of a diagnosis, or in other words the possible faults. Solving the following propositional Horn clause abduction problem thus allows us to derive possible explanations for the observed behavior.

Definition 5. *[9] Given a knowledge base KB* $= (P, \mathit{HYP}, \mathit{TH})$ *and a set of observations OBS* $\subseteq P$*, the tuple* $(P, \mathit{HYP}, \mathit{TH}, \mathit{OBS})$ *describes a propositional Horn clause abduction problem (PHCAP). A set* $\Delta \subseteq \mathit{HYP}$ *is a solution to a PHCAP, if and only if* $\Delta \cup \mathit{TH} \models \mathit{OBS}$ *and* $\Delta \cup \mathit{TH} \not\models \bot$*. A solution* Δ *is parsimonious or subset-minimal, if and only if no set* Δ' *such that* $\Delta' \subset \Delta$ *is a solution.*

Formally, a solution Δ of a PHCAP is a set of hypotheses that allows to derive the given observations via *TH*. Consequently, Δ is indeed an explanation of the given observations and we thus refer to Δ also as *abductive diagnosis* or simply as diagnosis. Please note that in practice, we are often only interested in subset-minimal diagnoses, so that in the literature, like in [2], we often find that a diagnosis has to be subset-minimal by definition.

"*Determining whether a hypothesis is included in a minimal diagnosis is NP-complete*" [18], where we would like to refer the interested reader to [20] for a comprehensive complexity analysis of logic-based abduction. If the set of hypotheses is not too large, we can compute the solutions efficiently though. Such an approach might use de Kleer's Assumption-based Truth Maintenance System (ATMS) [21; 22] and a newly generated proposition σ, encoding the observations as a single rule $o_1 \wedge \ldots \wedge o_k \rightarrow \sigma$ for $k = |\mathit{OBS}|$. 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 [23].

In the initial concept proposed in [8], we used only single fault activations. While this has the advantage of limiting the simulations to $(|\mathit{MODES}| - 1) * |\mathit{COMP}| + 1$ per input scenario (a test case as of Definition 1) in the worst case, it also means that fault interactions would not be considered in the abductive diagnosis model. Covering all combinations exhaustively would mean $(|\mathit{MODES}| - 1)^{|\mathit{COMP}|} + 1$ simulations per input scenario though, which is infeasible in practice. That is, let us assume that we have 20 components with 10 modes, and 100 input scenarios—which is not much since even for non-sequential behavior 10 variables with 10 possible values would result in 100 combinations. Then we would need $10^{20} * 100 = 10^{22}$ simulations, which would take about $3.17 * 10^{11}$ years to conduct if a single simultation takes about 1 millisecond (which is quite optimistic).

Inspired by the success of combinatorial testing, and its successful adaption to testing self-adaptive systems (and thus, in principle, the diagnosis engine underlying the adaption mechanism) as investigated by Wotawa in [24], we proposed in [9] the use of a combinatorial approach at conquering the parameter and signal/variable space for the individual simulations. While global exhaustiveness is not within reach, there the exhaustiveness focuses on the local interactions between variables. In particular, if we desire a *combinatorial strength* of s, this means that every s-way interaction between variables shall be considered in at least one simulation (or a test case in the context of combinatorial testing and Def. 1). Thus for every variable subset of size s, we have that every combination of values that we can assign to these variables is indeed part of at least one simulation. If s is equal to the number of variables this would translate to global exhaustiveness, but usually we have $s << n$ or at least $s < n$. The resulting scalability improvements allowed us to consider also injecting multiple faults in [9].

In the following table we briefly illustrate the concept for 3 Boolean health state variables h_i (one nominal mode, one fault mode), where we can easily see that simulations 1 to 4 would achieve a combinatorial strength of 2. Adding simulation 5, despite introducing a new value combination and thus increasing coverage in principle, is unnecessary to achieve $s = 2$. Furthermore, we could not retain $s = 2$ when replacing any of the simulations 1 to 4 with simulation 5. That is, when replacing simulation 1, $h_2 = h_3 = \bot$ would be missing, and when replacing simulations 2,3, or 4, $h_1 = \bot / h_2 = \top$, $h_1 = h_3 = \top$, or $h_1 = h_2 = \top$ would be missing. Together with simulations 6 to 8, simulation 5 can achieve a combinatorial strength of 2 though.

	h_1	h_2	h_3
simulation 1	$\bot$	$\bot$	$\bot$
simulation 2	$\bot$	$\top$	$\top$
simulation 3	$\top$	$\bot$	$\top$
simulation 4	$\top$	$\top$	$\bot$
simulation 5	$\bot$	$\bot$	$\top$
simulation 6	$\bot$	$\top$	$\bot$
simulation 7	$\top$	$\bot$	$\bot$
simulation 8	$\top$	$\top$	$\top$

The optimization potential that we exploit in a combinatorial approach is that we can cover more than one individual s-way interaction in a single simulation, which allows us to reduce the total number of required simulations. In our table above, a single simulation covers three two-way interactions for the three variable subsets of size $s = 2$. As we showed, the way how we overlay the various two-way interactions is crucial though, in order to achieve some desired strength with as few simulations as possible.

When deploying the technique, an immediate question is which strength we would need to target in practice. Kuhn and colleagues showed in [25] that 6-way strength might suffice to reveal all faults in a testing context, where for some domains the limit was even lower. Limiting the strength's scope to fault combinations, this would match the practice that we often limit the search for diagnoses in terms of cardinality (e.g. stop the search for triple-faults) under the assumption that it is unlikely that more than x components fail simultaneously. For the input signals and system parameters, it is less clear which strength would be needed for our task. So while there are several questions to be explored via empirical experiments in this respect, we surmised in [9] that the strength $s = 6$ identified by Kuhn et al. for testing might be an interesting starting point also for our diagnostic setting. That is, both settings are motivated by the aim to explore the behavior/exercise the system. The *coverage* achieved for software programs when using combinatorial testing shows also promising results in this respect [26].

If we have variables with varying domains, we can use *mixed-level covering arrays* (MCAs) to derive simulation settings as offered in the table for Boolean variables. It allows us to consider variable sets with individual domains (relating to the alphabets in our context) for each variable.

Definition 6. *A mixed-level covering array MCA$(V, (A_1, \ldots, A_k), s)$ of strength s for $k = |V|$ variables with their individual finite alphabets A_i is a two-dimensional $k \times n$ array such that for any $V' \subseteq V$ such that $|V'| = s$ we have that every combination in the cross product of the individual alphabets of the variables in V' appears in at least one of the n rows.*

When deploying an MCA algorithm we have to follow a three step process. First, we have to describe the set of variables V and the variables' individual alphabets A_i. These alphabets might contain a set of exemplary values (for a discussion of task-dependent qualitative abstraction see [13]), or a set of exemplary sequences for sequential behavior as we mused in [9]. In the context of combinatorial testing, this first step is coined input parameter modeling and we refer the interested reader to [19] for an introduction. The second step is to invoke a combinatorial decision procedure [27; 28], e.g., using a combinatorial test generation tool like ACTS [29], in order to generate an MCA as of Definition 6. In the third and final step, we take each individual row in the MCA and consider it as the input data for a simulation.

As we reasoned in [9], a combinatorial exploration concept might allow us thus to structurally (but necessarily incomplete in a global sense) attempt to representatively cover a system's correct and faulty behavior in our simulations. While we suggested to provide an alphabet of individual scenarios for the various input signals (variables) or a total alphabet of scenarios for the entire system, the task was left to the user. In this paper we investigate the automation of this step, which we start in the next section by discussing an appropriate input model.

3 Modeling the Input Signal Scenarios

In our definition of a system model (see Definition 1), we assumed that we have a simulation model available. This model allows us to simulate the system's behavior for a finite set of points in time, if the system's input signals as well as the desired fault modes are specified. Our aim is now to generate an MCA as of Definition 6 where the rows will define the individual simulation configurations (input signals, system parameters, fault modes) that will allow us to come up with an abductive diagnosis model as of Definition 5 via the algorithm described in [9].

In [9] we suggested that a user should define for each input signal a finite set of sequential behavior scenarios in order to represent the system's temporal behavior in a finite alphabet. This way one can handle input signals like standard variables when creating the MCA. Incompatibilities between the individual alphabets could pose problems when using the MCA generation tool though. That is, some simulations could be impossible as outlined in the introduction, which could either be a problem in the tool (if this is checked - see our later discussion of constraints), or if unchecked would result in a degraded combinatorial strength since the derived strength considered the entire set of simulations and not only the viable ones that we can indeed conduct. As an alternative solution, we thus suggested to define complete input scenarios (with a corresponding alphabet). This is certainly not an easy task by itself, and especially so, if we would like an approach to it to follow the local variable coverage idea behind the combinatorial exploration concept.

Let us start by defining a model that would enable us to automatically construct such an alphabet.

Definition 7. *Let I be a set of input signals, P be a set of system parameters, and O be a set of output signals. Each signal $i \in I$ and $o \in O$, as well as each $p \in P$ has its own finite alphabet A_i,A_o, or A_p respectively. An I/O model of a system is a tuple $\Phi = (\Sigma, Q, q_0, \rho)$ such that Σ is a finite alphabet defined as the cross-product of the individual alphabets for all $i \in I$, $o \in O$, and $p \in P$, Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\rho : Q \times \Sigma \to Q$ is a deterministic transition function that gives us the next state when reading some letter $\sigma \in \Sigma$ in some state $q_i \in Q$.*

It is obvious that our definition follows the concept of finite state machines (automata / symbolic transition systems), but we do not define any acceptance condition (finite or infinite). Viable transitions are defined by a function returning a single state, so that there is no non-determinism or alternation (the latter would allow to proceed to multiple states simultaneously). Since we do not require an infinite acceptance condition (we could assume a finite acceptance condition and that all states are accepting), simple subset-construction would allow us to translate any alternating definition to a deterministic one as of Def. 8 though, so that we can take advantage of the simpler deterministic definition without loosing generality.

In principle, this model implements a designers' temporal (sequential) reasoning about the system's *interfaces*, where we often use such models for a digital circuit's sequential behavior. Abstraction, as described before, allows us to use this concept also for the more general systems as allowed by Def. 1. In our simulation configurations, i.e., the test cases as of Def. 1, we do not consider any outputs. In order to obtain the system's *input language* (see Def. 9), we could thus ignore the values for all outputs in O for the I/O model's language as defined by the model's set of finite or infinite sequences. In terms of our model, simply removing these parts of $\sigma \in \Sigma$ for the "edge labels" could however result in non-determinism in the transitions. That is, the labels of two outgoing edges for some state q might differ only in the output signals. Technically, subset construction could be used to obtain a deterministic transition function automatically. From a system designer's perspective, we'd like to note though that this would describe a situation such that the system output would be determined non-deterministically—which *might* not have been the designer's intent.

Whether we construct the following input model defining a system's input language directly, or derive it from Φ might be different for each project. Nevertheless, we have that the two are obviously connected and that the necessary information to come up with it should be available in a design process. That is, the system *interface* has to be defined for a description how to use it (the input part at the least), and when developers decide which components should interact which way in order to implement the desired functionality, also the component interface data should be available.

Definition 8. *Let I be a set of input signals, and P be a set of system parameters. Each input signal $i \in I$, as well as each $p \in P$ has its own finite alphabet A_i or A_p respectively. An input model of a system is a tuple $\Phi_I = (\Sigma_I, Q_I, q_0, \rho_I)$ such that Σ_I is a finite alphabet de-*

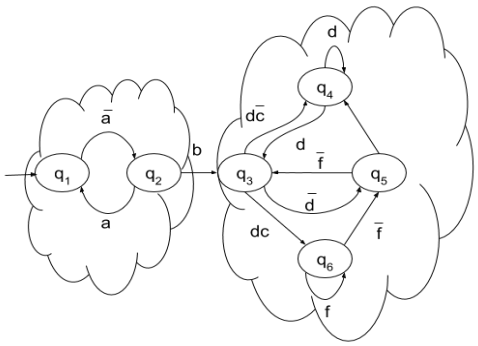

Figure 1: An input model with two SCCs for Boolean signals $\{a, b, c, d, f\}$, where $\bar{a}$ denotes "not a", and edge label a means that we take this if a is true for $\sigma \in \Sigma_I$.

fined as the cross-product of the individual alphabets for all $i \in I$ and $p \in P$, Q_I is a finite set of states, $q_0 \in Q_I$ is the initial state, and $\rho_I : Q_I \times \Sigma_I \to Q_I$ is a deterministic transition function that gives us the next state when reading some letter $\sigma \in \Sigma_I$ in some state $q_i \in Q_I$.

Definition 9. *A system's finite or infinite input language is the set of possible finite or infinite sequences of letters $\sigma \in \Sigma_I$, such that starting from the initial node q_0 there is always a transition from the current node $q \in Q_I$ to another node $q' \in Q_I$ defined by ρ_I for the current letter (the ith one) in the sequence, when considering the sequence letter by letter.*

An input model as of Def. 8 consists of the input sequences allowed by a system. Interpreting it as an automaton with a finite acceptance condition such that the set of final states is equal to Q_I, we can formally derive a corresponding finite input language as of Def. 9. Such language data can be of advantage when creating the MCA, since we then could avoid constructing impossible simulation configurations without sacrificing the internal reasoning regarding the combinatorial exploration (see also Section 4).

When we consider an input model's properties in detail, there arise several further questions of interest. For instance whether we could exploit the input model's *structure* in the MCA generation. That is, not only the language itself, but also the model's structure. So, if we discover an input model's *strongly connected components* (SCCs, see Def. 10), we basically can determine language fragments that in their possible sequences define the input language. This might unveil that several variable combinations can be achieved only in certain SCCs and might thus be incompatible with others. Considering reachability in terms of these SCCs would then allow us to identify necessary chronology (like when we have to go through SCC A when aiming for a specific combination present only in SCC B), or mutual exclusiveness (when we cannot visit SCCs D and F in the same simulation). Obviously, all these data could help in the process of creating the MCA.

Definition 10. *Let Φ_I be an input model as of Def. 8. A* strongly connected component *in Φ_I is a maximal set of states $Q' \subseteq Q_i$ such that for any two $q_i, q_j \in Q'$ there is a (possibly empty) sequence of individual letters $\sigma \in \Sigma_I$ such that we can reach q_j from q_i via ρ_I and this input sequence.*

4 Covering the Input Behavior

Instantiating the signals for each time step and thus establishing individual variables certainly allows us to use an MCA for covering temporal behavior. As we surmised in the last section, we should however also think about variable value incompatibilities implementing, for instance, physical impossibilities. Luckily enough, MCA tools like ACTS [29] allow us to specify constraints that the individual combinations must adhere to. Thus, in principle, we can indeed accommodate such reasoning by feeding corresponding constraints derived from our input model to the tool. In principle we would have to unroll the input model Φ_I for the desired simulation length and encode it in constraints. Constraint support is limited to selected algorithms though, like to IPOG and IPOG-F for ACTS version 2.92. While, e.g., IPOG [27] uses internally a combination enumeration in its greedy strategy—where we can then restrict also these enumerations—for algorithms like IPOG-D [27] it is less clear how to effectively implement such a reasoning step. Simply creating solutions to be tested and possibly discarded afterwards is not ideal, since we're likely to have a lot of constraints and discarding a solution might affect the currently achieved strength also for other variable subsets.

The temporal unrolling principle is similar to how we would unroll a formal property when creating a satisfiability encoding for a test oracle as outlined in [30], or that for bounded model-checking proposed in [31]. It is important to note though, that we have to prevent sequences that do not *strictly* follow ρ. This means that we have to prohibit transitions from any state q_i with some letter $\sigma \in \Sigma_I$, for which q_i has no outgoing transition specified. A *relaxed* encoding would only encode constraints implementing forward reasoning, that is, expressing that if we are in state q_i and read letter σ we move to state q_j. This can however result in the encoding allowing also other transitions *simultaneously*, like it is possible in an alternating transition relation. In some applications this might help, like when we could reduce the length of counterexamples for our experiments with symbolic implementations of alternating automata in [32]. In our case it would be counterproductive though. In order to address this, it suffices to state that we proceed to q_j *only*, since we require ρ to be deterministic. How to encode this exactly will depend on the actual encoding, but it is apparent that we would require some "current state information" for each time step. This could mean to use an additional variable per time step with the domain of Q_I. Strictness would be ensured, since then the presence in states $q \in Q_I$ would be mutually exclusive by default. For an alternating transition relation/function, and/or individual state variables, this would be more complicated.

In a more direct approach that takes the input model as further input, we could avoid state variables for Q_I in the MCA. As suggested in the last section, we could then also exploit the various SCCs' language fragments (and thus the structure of Φ_I's input language) when identifying new combinations in the combinatorial exploration. This requires the development of a new MCA engine though. While we are indeed currently investigating options to do this in the context of algorithms like IPOG, such a solution is currently not available (nor is some internal prototype). Such a dedicated solution would have also other advantages over an off-the-shelf MCA solution, like that we would not need to formally translate Φ_I to a constraint representation.

In respect of advantages, let us briefly consider the *scope* of the local exhaustiveness concept. When considering temporal behavior, in general we have that variable values might interact with each other independent from the time when they are assigned. Of course they can also be completely independent from each other. Some of the information about such interactions is contained in the I/O model (edge-labels, SCC strucure, etc.). While we already might lose some information about this in Φ_I (vs. Φ), the individual SCCs like in Figure 1, for instance, still tell us that the variables labeling its edges do interact. Please note that the edge label format chosen for Fig. 1 is not only more succinct (possibly exponentially) than creating an individual edge for each single letter, but it also shows us the *important* variables for the individual transitions. That is, all variables not in the edge label are ignored for this transition (can be assigned any value). Such "free" variables for a transition certainly offer a lot of potential for overlaying other variable interactions, but it is questionable what information is gained then. That is, they are meaningless for this transition, and might be meaningless also in terms of the system at this point in time or the whole execution. If we consider, e.g., some edge labeled $\bar{a}$ and combine it with the combination dc, for $s = 2$ the MCA generation algorithm would consider dc to be covered and would not require it to reappear again. The problem is however that taking an edge labeled dc *might* reveal more problems, as could some sequence where we assign d at one time step and c at another one (but they do interact). This simple example shows us several things:

- While the input model highlights some variable interactions that we should check, some are not visible (e.g. data dependencies). Still, we should take any available data into consideration when generating the MCA, ideally via an interface for optional input data.
- The temporal scope of the MCA algorithm's considerations might not always be the one we aim at. Sometimes we might prefer interactions between SCCs, sometimes within an SCC (but at different time steps), sometimes at a specific time step. In practice, a combination might be called for, where different strengths might suffice and would allow us to restrict the search.

From a more formal perspective, the MCA will feature $|I| * |\mathit{TIME}|$ columns, one for each input signal i for time step $t \in \mathit{TIME}$. A combinatorial approach with strength 2 will not only try to cover each variable value combination for each time step and variable subset of size 2, but will try to do so also for variable subsets such that the individual variables belong to different time steps. If we assume that all variables have the same domain with m (abstract) values, the simulations would have to cover $\binom{|I|*|\mathit{TIME}|}{s} * m^s$ individual combinations for the various variable subsets (if they are allowed by the input model). For $|I| = 20$ input signals with $m = 10$, a sampling rate of 1 kHz and $t_e = 1$ s, this would mean about $8.88 * 10^{28}$ combinations, to be heavily filtered by the constraints as defined by the input model.

Let us now think about focusing the combinatorial exploration on a single time step. Instead of covering all $m^{|I|} = |\Sigma_I| = 10^{20}$ letters in the input alphabet, this would mean to cover $\binom{|I|}{s} * m^s$ partial assignments, where this number is $1.14 * 10^6$ for a combinatorial strength of 3 and the example of above. If $|I|/s \geq 2.0$, we can indeed overlay also some of these partial combinations in a single simulation, and we would like to note that again we have that the input model might allow to decrease this number significantly. If we would like to have all possible partial letters simulated for each time step, then some upper bound we can derive is the product of the given number and the number of time steps, if we would naively create one simulation for a single partial assignment and time step ignoring the optimization potential (which would depend on the input language to some extent).

So far, we have been assuming that we would know how long a simulation should be. A simple lower bound for this length could be determined via the following definition of *minLen*(Σ) and could help to avoid unfortunate guesses.

Definition 11. *Let Φ_I be an input model as of Def. 8 and let $L(\Phi_I)$ be its input language as of Def. 9. The minimal length of a simulation that is able to produce some letter $\sigma \in \Sigma_I$, coined minLen(σ), is the length of the shortest sequence of letters in $L(\Phi)$ s.t. its last letter is σ. The minimal length of a simulation to possibly produce any $\sigma \in \Sigma_I$, coined minLen(Σ_I) is the maximum of minLen(σ) over Σ_I.*

A more complex variant of this definition might not focus on letters, but variable assignments within letters. Other procedures to compute the minimal length (which is directly related to the breadth of the MCA) could consider SCC interactions and other aspects. Please note that it is important to compute such a minimal length, since when guessing it, the constraints from the input model might just prohibit the MCA algorithm from creating a variable combination (if the value combination is impossible to achieve within the limits, i.e., in the reachable part of the input model). In particular, we would not get the feedback that it would be possible to have this combination when considering longer sequences.

Since in practice the available resources are limited, it might not always be necessary, and in our interest, to create simulations of the same length. For single simulations, we might require only some shorter versions in order to achieve the same desired strengths. If we focused on covering variable assignments not individually for each time step, a post processing step could optimize the simulations as suggested by the MCA to some degree by shortening them taking this into account. Obviously, this is another reason why a dedicated MCA generation algorithm, or a combinatorial exploration algorithm tailored for our cause could be quite attractive, i.e., when we accommodate options in its interface to consider such aspects.

Summarizing, we can see that, in principle, we can indeed apply a combinatorial exploration approach based on standard techniques to cover also the input space, based on an input model as defined in Def. 8. We showed in our discussions though, that a solution tailored towards our task could offer various advantages in respect of performance and control over the search. We also showed that, while a combinatorial exploration is in principle an uneducated approach, we can exploit structural data in order to optimize the result. This concerns, for instance, the construction of single simulation configurations where we could consider the reachability and language fragments of an input model's SCCs. Some discussed optimizations could also improve the meaning of overlaid variable interactions, when taking do-not-care situations and varying temporal scopes of the combinatorial idea into account. As mentioned, this might include also the use of varying strengths for different variable subsets (or in the scope of different SCCs).

	$\lvert\Sigma_S\rvert = 100$											
	$k = 3$						$k = 6$					
	$n = 20$			$n = 50$			$n = 20$			$n = 50$		
t_s	1 ms	1 s	1 min	1 ms	1 s	1 min	1 ms	1 s	1 min	1 ms	1 s	1 min
t_t	31.7 h	3.62 y	217 y	123 y	$123 * 10^3$ y	$7.37 * 10^6$ y	22.7 d	62.2 y	$3.72 * 10^3$ y	$5.04 * 10^4$ y	$5.04 * 10^7$ y	$3.02 * 10^9$ y

Table 1: The total runtime for the simulations as estimated by Def. 12 in hours (h) days (d) or years (y).

Especially the latter is of interest for our decision of whether we would like to generate the input scenario alphabet first and use the concept proposed in [9], or whether we would prefer a single MCA algorithm invocation considering all variables (input, mode assignment, parameters) concurrently. If we construct Σ_S first, we can approximate the time to run all simulations as follows:

Definition 12. *Given a system with n components with m modes each, a set Σ_S of input scenarios, a maximum diagnosis cardinality k, and the average simulation time t_s of a single simulation, we can estimate the time t_t to simulate each input scenario for each fault scenario allowed by k with the formula $t_t = \binom{n}{k} * m^k * t_s$.*

For the example from Section 2 with $n = 20$, $m = 10$, and $|\Sigma_S| = 100$, let us assume that we limit our exploration to diagnoses (fault combinations) of up to size $k = 3$. Let us also investigate $n = 50$ for a larger (but still small) system, and $k = 6$ in order to match the comb. strength suggested by Kuhn et al. . In Table 1 we can see t_t for $t_s \in \{1ms, 1s, 1min\}$. From these figures, it becomes clear that we might be able to come up with the necessary resources for some projects (31.7 hours for $n = 20, k = 3, t_s = 1$ ms), but not for all (t_t becomes 123 years even if just changing n to 50). Thus we might *need* to cover also the fault space with an isolated combinatorial approach, or use a single MCA algorithm invocation considering the whole variable space. Of course this will affect the diagnostic accuracy of the resulting diagnostic model [9]. That we can consider multiple strengths also with current tools like ACTS, provides the necessary background to explore such options and their compromises in respect of the resulting abductive model's efficiency and the resources needed to create it.

5 Related Work and Discussion

Most of the techniques we draw on have been available for some time and have been used in other contexts. This includes the input model in the form of some finite state machine, like, e.g., used in model-checking [31]. Also there, SCC analyses might be exploited in one or the other way [33]. Combinatorial exploration [11; 12; 19] and algorithms like IPOG and IPOG-D [27] for creating MCAs have been proposed in the context of combinatorial testing. The contributions of this paper thus do not lie in the basic techniques themselves, but in the concept that draws on them for creating an abductive diagnosis model.

To the best of our knowledge, there has not been much work with such a focus (see [8; 9]). Nevertheless, and as has been discussed, we can profit from a lot of research in the context of combinatorial testing, which is a very active research area. Aside technical improvements like new algorithms or surveys [28], also recent investigations like [26] are of interest, since they focus on establishing a connection between traditional coverage methods like Modified Condition/Decision Coverage (MC/DC) and combinatorial (testing) concepts. The consideration of such connections will allow us to make more educated guesses about our computations' efficiency, the parameters we have to consider, the impact additional (secondary, structural) data might have, as well as the efficiency of the resulting diagnostic model. So far, the findings offered by related work support the validity of our idea. Future research and experiments with various implementation options as discussed in the paper, will have to confirm practical viability though.

6 Conclusions

In this paper we proposed the basis for covering the input signal behavior in the context of creating an abductive diagnosis model via a workflow based on fault injection, simulation, and combinatorial exploration that we initially proposed in [9]. We proposed a corresponding model, discussed several arising issues specific to the coverage of temporal input behavior, offered options for considering auxiliary data in order to improve the computation and the result, and showed that we can, in general, indeed use a model to automatically derive the desired input scenario coverage with available techniques and tools.

We also showed though that a dedicated solution could offer significant advantages, and explored some corresponding directions for future research. That is, in our discussions we unveiled potential options to flexibly consider also secondary data (if available), allowing us to offer options for a flexible fine-tuning of the result and its computation.

In the context of our earlier work, the research in progress as described in this paper provides ample room for future work. That is, we will explore implementations of the proposed concepts, and corresponding experiments shall investigate the effects of all the individual parameters. While our discussions offer a variety of future research directions, they also showed that the needed efforts might be well spent due to the potentially achievable results that could certainly help with the adoption of diagnostic reasoning in practice.

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

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

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

[4] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *International Conference on Software Engineering (ICSE'02)*, pages 467–477, 2002.

[5] R. Abreu, P. Zoeteweij, and A. J. C. v. Gemund. Spectrum-based multiple fault localization. In *2009 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 88–99, Nov 2009.

[6] B. Hofer and F. Wotawa. Spectrum enhanced dynamic slicing for better fault localization. In *European Conference on Artificial Intelligence (ECAI'12)*, volume 242 of *ECAI*, pages 420–425, 2012.

[7] M. Weiser. Programmers use slices when debugging. *Comm. of the ACM*, 25(7):446–452, July 1982.

[8] B. Peischl, I. Pill, and F. Wotawa. Abductive diagnosis based on Modelica models. In *27th Int. Workshop on Principles of Diagnosis (DX)*, 2016. `http://dx-16.org/papers/DX-2016_5.pdf`.

[9] I. Pill and F. Wotawa. Model-based diagnosis meets combinatorial testing for generating an abductive diagnosis model. In *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 248–263, 2018.

[10] 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.

[11] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. 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. Sachenbacher and P. Struss. Automated qualitative domain abstraction. In *International Joint Conference on Artificial Intelligence*, pages 382–387, 2003.

[14] Timothy Budd, R. DeMillo, R. Lipton, and F. Sayward. Theoretical and empirical studies on using program mutation to test the functional correctness of programs. In *Proc. Seventh ACM Symp. on Princ. of Prog. Lang. (POPL)*. ACM, January 1980.

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

[16] I. Pill, I. Rubil, F. Wotawa, and M. Nica. SIMULTATE: A toolset for fault injection and mutation testing of simulink models. In *IEEE International Conference on Software Testing, Verification and Validation (ICST) Workshops*, pages 168–173, 2016.

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

[18] G. Friedrich, G. Gottlob, and W. Nejdl. Hypothesis classification, abductive diagnosis and therapy. In *First International Workshop on Principles of Diagnosis*, 1990. Also appeared in Proc. of the Int. Workshop on Expert Systems in Engineering, LNAI Vol.462, 1990.

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

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

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

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

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

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

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

[26] D. Li, L. Hu, R. Gao, W. E. Wong, D. R. Kuhn, and R. N. Kacker. Improving MC/DC and fault detection strength using combinatorial testing. In *IEEE Int. Conf. on Software Quality, Reliability and Security Companion (QRS-C)*, pages 297–303, 2017.

[27] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence. IPOG-IPOG-D: Efficient test generation for multi-way combinatorial testing. *Software Testing, Verification, & Reliability*, 18(3):125–148, 2008.

[28] Victor Kuliamin and Alexander Petukhov. Covering arrays generation methods survey. In *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 382–396, 2010.

[29] 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.

[30] I. Pill and F. Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139:124 – 141, 2018.

[31] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Alg. for the Construction and Analysis of Systems*, pages 193–207, 1999.

[32] R. Bloem, A. Cimatti, I. Pill, and M. Roveri. Symbolic implementation of alternating automata. *Int. J. of Foundations of Comp. Science*, 18(04):727–743, 2007.

[33] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, July 2001.