Exploiting Observations from Combinatorial Testing for Diagnostic Reasoning

Ingo Pill\(^1\) and Franz Wotawa\(^1\)

\(^1\)Christian Doppler Laboratory for Quality Assurance Methodologies for Cyber-Physical Systems
Institute for Software Technology, Graz University of Technology

e-mail: {ipill, wotawa}@ist.tugraz.at

Abstract

Assessing the correctness of a system is not an easy feat and usually we use a variety of techniques and tools in such a process—ranging from formal methods via testing concepts to diagnostic reasoning. In this manuscript, we discuss how to integrate combinatorial testing and model-based diagnosis, where we aim to exploit data obtained from multiple behavioral observations derived via combinatorial testing in order to characterize evident problems in the considered system. In particular, we combine the conflicts obtained for all failed test runs, and discuss the effects achieved in this non-monotonic diagnostic process.

1 Introduction

Getting your design right is often quite a challenge, and aside following design patterns that assist us in our development steps, we usually draw on a variety of techniques and methods to finally derive a system as close to perfection and correctness as possible. Formal methods like model-checking \(^1\) allow us to prove properties of a system model, testing concepts and corresponding test oracles \(^2\) allow us to consider and evaluate exemplary system behavior (for simulations and real hardware as well as software), and diagnostic reasoning \(^3\) allows us to investigate what went wrong if we find that a system does not behave as expected.

Despite the fact that diagnostic reasoning can help designers in the process of isolating the source of encountered misbehavior, in practice, developers scarcely rely on corresponding tools. While this might stem from several reasons like the lack of a diagnostic model, sometimes it is just that the benefits and possible integration with currently deployed techniques are unknown. In this manuscript, we will focus on combining combinatorial testing \(^5\) with consistency-oriented model-based diagnosis as described by de Kleer and Williams \(^4\) or Reiter \(^3\) in the 80ies.

That there is a lot of potential for combining diagnostic reasoning and combinatorial testing is evident also from other work. We have been discussing, for instance, ways to exploit the exploration concept behind combinatorial testing for generating an abductive diagnosis model \(^6\), resulting in a promising concept for coming up with a knowledge-base representative enough for abductive diagnostic reasoning. In contrast to our earlier work, we are focusing on consistency-oriented diagnostic reasoning in this manuscript. While we can certainly use the approach proposed in Section \(^4\) as benchmark for an evaluation of the abductive concept from \(^6\), an imminent advantage of the proposed concept is that we draw for our diagnostic reasoning process on a system model (and not a derived knowledge base) that allows us and a user to explain (and in an abstract sense prove) a computed result in relation to this behavioral system model. Especially in safety-critical domains like the automotive industry, this can be an attractive aspect increasing the trust in the diagnostic process.

Furthermore, we aggregate data from all failed test runs for a test suite constructed in the context of combinatorial testing \(^7\), and consequently exploit all available data about a system’s behavior in our reasoning. This is in contrast to the usual consistency-oriented diagnosis scenario, where we are interested in explanations for some specific exemplary behavior. That aggregating data from multiple behaviors (or in other words for multiple different stimuli) results in non-monotonic reasoning is well known \(^3\), and can be intuitively deduced from the fact that individual faults will not always result in effects observable for a specific scenario. This is opposed to the monotonic approach of making additional measurements for the same scenario, like when we take additional measurements in an electronic circuit for the same input stimulus. While we do not compute new distinguishing tests for discrimination purposes \(^6\), nor discriminate between diagnoses derived for a diagnosis problem via data observed for new tests, we compute diagnoses taking all the data observed for a combinatorial test suite into account. With this focus we draw on the aspect that such a test suite was specifically constructed to achieve a good behavioral coverage by taking component interactions into account (see Section \(^2\)) rather than focusing on exercising the set of individual components.

In particular, we collect the conflicts we derive for diagnosing the individual failed test cases up to a given cardinality limit (for the diagnoses), and then compute the minimal hitting sets for those conflicts. These minimal hitting sets coined multi-observation diagnoses (see Def. \(^7\)) shall serve as our characterization of the problems present in the system. While obviously also combinatorial testing is not able to cover a system’s entire behavioral space, the underlying assumption behind our choice is that the resulting approximation is of high quality due to the specific concept. An interesting aspect there is that we can choose the desired preciseness (impacting the test suite size) via the value chosen for the combinatorial strength (see Section \(^2\)).

In terms of our presentation, we will discuss the basics...
of consistency-oriented model-based diagnosis and combinatorial testing in the next section, prior to introducing our working example in Section 3. After proposing the details of our approach in Section 4, we show first experiments for our working example, and discuss related work before concluding with a summary and final remarks.

2 Of combinatorial testing, conflicts, hitting sets, and model-based diagnosis

When testing a system, we exercise it for specific input scenarios as defined in the individual test cases of the employed test suite, with the aim of unveiling faults in the system. Since it is obvious that we can’t exercise the system for every possible input scenario (think of the $2^{32} \times 32$ input scenarios for a 32-bit adder that would take about 584 million years to execute on a single machine if executing one test takes a millisecond), combinatorial testing was invented to structurally cover the viable input variable combinations. The underlying idea is to cover all s-way combinations of variable values in at least one test-case if it is not possible to cover all variable combinations for a globally exhaustive approach. This parameter s is referred to as the combinatorial strength of a test suite and defines the scope of the local exhaustiveness. If we would choose s to be equal to the number of variables n, then we would indeed cover all possible variable values resulting in total coverage, i.e., global exhaustiveness. Usually we choose some s << n (or at least s < n) though, and thus focus on covering all value assignments for every variable subset of size s in at least one test case. The efficiency of such an approach thus heavily depends on how many such combinations we can cover in each individual test case, i.e., how many test cases are needed to achieve the desired combinatorial strength.

In our setting, we do not need to define the expected output for a test case since we can judge correctness via the system model used for the diagnostic reasoning process. For highlighting the connection to combinatorial testing, we will still use the definitions of a test case and a test suite though.

Definition 1 (test case). A test case $t = (\delta, \rho)$ for some system $S$ with inputs I and observable outputs $O$ defines an input stimulus $\delta$ for $S$ via defining desired values for all inputs $i \in I$. The expected output is defined in $\rho$ either via defining the expected values for all $o \in O$, or via some function for implementing an automated test oracle [3].

Definition 2 (test suite). A test suite $T$ is a finite and non-empty set of test cases as of Definition 1.

With combinatorial testing we aim to derive a test suite $T$ that covers the input space with the desired combinatorial strength. Before showing how to derive $T$ via mixed-level covering arrays (see Def. 3), let us illustrate the combinatorial testing concept for a simple example. Let us assume that we have a simple combinatorial circuit (like the one shown in Section 2) with three Boolean input signals a, b, and c so that we have $n = |I| = 3$. While total coverage of the input combinations is within reach, since $2^3 = 8$, let us assume that we aim for a combinatorial strength of $s = 2$.

In the table below, where we denote true with $\top$ and false with $\bot$, we have all eight possible test cases that would allow us in their union to define a globally exhaustive test suite. We can also see that we can achieve a combinatorial strength of 2 with four test cases only, for example with $T = \{t_1, t_2, t_3, t_4\}$ and also with $T' = \{t_5, t_6, t_7, t_8\}$.

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td></td>
<td>$\top$</td>
<td></td>
<td></td>
<td></td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\bot$</td>
<td>$\bot$</td>
<td></td>
<td>$\bot$</td>
<td></td>
<td>$\bot$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\bot$</td>
<td>$\bot$</td>
<td></td>
<td>$\bot$</td>
<td></td>
</tr>
</tbody>
</table>

Please note that there can be no smaller $T$ for $s = 2$, since we need to cover $2^s = 4$ different value assignments for any individual Boolean variable subset of size s. Considering $T$, we can also see that adding $t_5$ to $T$ would increase coverage in principle via introducing a new value combination, but we would not increase the combinatorial strength with its specific focus regarding coverage.

We can furthermore see that we would not retain $s = 2$ when replacing any of $T$’s four test cases with $t_5$. That is, when replacing $t_1$, we would lack the combination $b = c = \bot$, and when replacing $t_2, t_3$ or $t_4$, we would not cover the combinations $a = \bot/b = \top$, $a = c = \top$, and $a = b = \top$ respectively. Together with $t_6, t_7$ and $t_8$, $t_5$ forms $T'$ of size four and with a combinatorial strength of 2 though. So, while it is easily seen that any of the test cases covers three two-way variable interactions for the three variable subsets of size 2 ($\{a, b\}, \{b, c\}, \{a, c\}$), the way we arrange them for the individual test cases is important for achieving the desired strength with as few test cases as possible.

The main question is now how to compute a test suite with a desired strength for some given set of variables. Since we often not only have Boolean variables, let us consider mixed-level covering arrays (MCAs) that allow us to consider variables with individual domains, i.e., finite alphabets representing individual finite sets of variable values.

Definition 3. 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 $k \times m$ 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 $m$ rows.

When deriving an MCA, we have to describe the set of variables I and the variables’ individual alphabets $A_i$ first. In our case, those are the system’s input values, and for our example we use Boolean variables only. For more complex (possibly continuous) variable domains, these alphabets might focus on a set of exemplary values though (for a discussion of task-dependent qualitative abstraction see [9]), where we discuss in [6] also the use of temporal input sequences. This initial step is referred to as input parameter modeling, where a corresponding introduction can be found in [7]. When I and all the alphabets $A_i$ are determined, we can proceed by invoking a combinatorial decision procedure [10, 11], e.g., by using the combinatorial test generation tool ACTS [12] for generating the desired MCA. Finally, we consider the MCA and let each of the $m$ rows intuitively define the stimulus $\delta_i$ for some test case $t_i$—such that we derive a test suite $T$ of size $m$.

A remaining question is now that of which combinatorial strength one should aim for. Investigating several problem domains, Kuhn and colleagues found in [13] that $s = 6$
As we discussed in the introduction, we’re aiming to combine the diagnostic data for all those tests in T that failed by showing unexpected behavior—with our aim being to characterize the faults present in the considered system and triggered by T. For generating the diagnostic data, we draw on the concept of consistency-oriented model-based diagnosis as characterized by de Kleer and Williams in [4] and Reiter in [3]. There, a system description SD describes the behavior of the system in sentences of the form \( h_i \rightarrow \text{nominal behavior of } c_i \). Those sentences state that under the assumption that some component \( c_i \) works correctly (encoded in a health state variable \( h_i \in H \)) we know how the component shall behave. We then have to provide such sentences for all components (all \( h_i \in H \) defining the set of health state variables), and complement them with the connections between the components and other system and background knowledge. Depending on the formalism, such background knowledge includes, e.g., the mutual exclusiveness of values for some signal. Since we thus make no assumptions about how the components behave in case of a fault, the approach is considered to implement a weak fault model. In principle, there exists also a theory for considering fault models, but it is out of the scope of this paper. Now, given some observations OBS about the system’s actual behavior, we can reason whether OBS is consistent with the expected behavior described in SD under the assumption that all components work as expected (SD \( \cup \) OBS \( \cup \{ h_i | h_i \in H \} \) is consistent or satisfiable). If this is not the case, we can furthermore define and verify hypotheses \( \Delta \subseteq H \) concerning faulty components [4][3][15][16] checking SD \( \cup \) OBS \( \cup \{ h_i | h_i \in H \setminus \Delta \} \) then.

**Definition 4.** A diagnosis for \( (SD, H, OBS) \) is a subset-minimal set \( \Delta \subseteq H \) such that SD \( \cup \) OBS \( \cup \{ h_i | h_i \in H \setminus \Delta \} \) is consistent (satisfiable).

We can compute the diagnoses as minimal hitting sets of the set of conflicts, where it suffices to focus on the subset-minimal conflicts [3].

**Definition 5.** A conflict \( C \) for \( (SD, H, OBS) \) is a subset of \( H \) such that SD \( \cup \) OBS \( \cup \{ h_i | h_i \in C \} \) is inconsistent (unsatisfiable). If no proper subset \( C' \) of \( C \) is a conflict, then \( C \) is a subset-minimal conflict.

**Definition 6.** A hitting set \( \Delta \) for a set CS of conflicts as of Def.5 is a subset of \( H \) such that \( \Delta \cap C_i = \emptyset \) for all \( C_i \in CS \). \( \Delta \) is a minimal hitting set if and only if no proper subset of \( \Delta \) is also a hitting set.

**Corollary 1.** A diagnosis \( \Delta \) for \( (SD, H, OBS) \) is a minimal hitting set for the set of all conflicts for this diagnosis problem. Since \( \Delta \) will hit also all non-minimal conflicts if it hits the minimal ones, it suffices to focus on the minimal conflicts only when computing the diagnoses.

As we will show in Section 3 for our example, we can easily derive a satisfiability problem for some combinatorial circuit in order to check the consistency of the observations with the system description, and there exist a multitude of manuscripts showing how to derive similar solver encodings for various application domains, e.g., for formal temporal descriptions in a temporal logic [17].

**3 A simple working example**

For our illustrations, we will use a simple combinatorial circuit with two AND-gates and one OR-gate as shown in Figure 1.

![Figure 1: A simple combinatorial circuit.](image)

Taking conflicts between the observed behavior and the expected one (as described in the system description) into account, we can easily derive the diagnoses using algorithms like RC-Tree [15]. Like RC-Tree, many algorithms can compute the conflicts on-the-fly by calling a theorem prover (like a SAT solver) to check the consistency of the observations with the system model for some diagnosis hypothesis (some \( \Delta \subseteq H \) as described above and ask the solver for a conflict set (a minimal unsatisfiable core in \( H \)) if this is not the case. Since we often limit our search for diagnoses in terms of cardinality, this can help to increase actual performance. That is, we might not need to compute all conflicts but only a subset as describing all \( \Delta \) s.t. \( |\Delta| \leq y \) if we’re interested only in diagnoses smaller than a given bound \( y \). This implements a common and intuitive assumption that we would consider it unlikely that more than \( y \) components fail simultaneously. Also our approach as proposed in Section 2 will take such a bound into account.
SD in conjunctive normal form (CNF):

\begin{itemize}
  \item \( \neg h_1 \lor \neg a \lor v \)
  \item \( \neg h_2 \lor \neg c \)
  \item \( \neg h_1 \lor \neg a \lor b \)
  \item \( \neg h_3 \lor \neg c \lor v \)
  \item \( \neg h_2 \lor \neg a \lor \neg b \)
  \item \( \neg h_3 \lor \neg a \lor \neg b \lor c \)
  \item \( h_2 \lor \neg a \lor \neg b \)
  \item \( h_3 \lor \neg a \lor \neg b \lor c \)
\end{itemize}

Our experiments in Sec. 3 will show that drawing on these nine clauses for describing the expected behavior of our circuit, and adding unit clauses catching the signal values from some observations OBS allows us to (a) check the consistency of OBS with SD in the scope of some theory about which gates work correctly or not (an assignment to all \( h_i \in H \)) and consequently to (b) derive corresponding diagnoses using the diagnostic theory described in Sec 2.

4 Aggregating diagnostic data from all failed test runs for a combinatorial test suite

The motivation behind our idea is quite straightforward. That is, computing diagnoses for observations concerning a single behavior of a system, e.g., for a single test execution, takes only the effects experienced for this execution into account. Of course we have to consider that some faults in the system might not result in any observable effect at all (e.g., if this component is not triggered for this specific stimulus), or multiple faults might interact in a way that we can not observe an effect, i.e., they mask each other. Intuitively, aggregating observations from multiple behaviors of a system should thus give us a better chance to indeed see some effects for the faults in a system (and corresponding interactions) that we can exploit for diagnostic reasoning.

The imminent question now is which example behaviors one might want to consider. Our suggestion is to build upon combinatorial testing in this case, which is an attractive test case generation concept following an intuitive motivation as outlined in Section 2. The reason for suggesting this is straightforward. For one, combinatorial testing as described in Section 2 was designed to test a system to the best of our knowledge, which is, of course, the case also for every other test case generation algorithm. What makes combinatorial testing special though, is that it is a structural approach that makes sure that all signal and variable interactions up to the certain desired strength \( s \) are considered in at least one testcase. In this respect, and also thinking now in terms of faulty components, this means that it is ensured that for a strength of 3, we have that all possible assignments to all possible variable subsets of size 3 are covered by a test suite. Coming back to the fault interactions considered above, this way, we can thus ensure to the best of our ability that certain fault interactions are indeed covered by \( T \).

In this context, and as briefly discussed in Section 2, modeling the input to an MCA algorithm used to derive \( T \) is an important step in generating some test suite via combinatorial testing. We might also want to tune this process in the sense that we might want use different strengths for individual variable subsets. So we discussed in 6 several aspects and options in respect of how to design and optimize a combinatorial behavioral exploration concept that we back then used for constructing a knowledge base extensive enough for abductive diagnosis. While the focus of our current work is different, it is related enough so that we would still like to refer the interested reader to those papers for our discussions of how one might want to optimize the input-modeling step for diagnostic purposes (if corresponding steps are possible). As expressed before, the motivation behind our current work is a bit different though.

In particular, like we outlined in the introduction, we focus on an integration with a testing process, assuming that that process was already deployed. Thus we have in turn to assume that \( T \) was constructed already.

In the following, we describe our approach of how to extract diagnostic data for coming up with a fault characterization based on all execution data obtained when executing \( T \). While we thus could assume the execution data to be already given, we still report the retrieval as Stage 1 in our Algorithm 1 in order to illustrate which data are needed for our concept. Since the same observations would be needed by a test oracle to arrive at a verdict for some test execution, one can assume though that the same data would certainly be produced in a standard testing process as well.

Algorithm 1 Computing diagnoses from the conflicts of all failed individual test case executions

**Require:** a combinatorial test suite \( T \), a system description \( SD \), a cardinality limit \( l \), and functions \( diag(OBS, SD, l) \) as well as \( MHS(CS, l) \) that compute the diagnoses and conflicts for a diagnosis problem \( OBS, SD, l \) for the former and the set of minimal hitting sets up to cardinality \( l \) for a set of conflicts \( CS \) for the latter.

```plaintext
1: procedure COMB DIAG(\( T, SD, l \))
2: \( T' \leftarrow \emptyset \)
3: \( CS' \leftarrow \emptyset \)
4: for \( t \in T \) do \( \triangleright \) Stage 1
5: \( (v, OBS) \leftarrow execute(t) \)
6: if \( v = \text{failed} \) then
7: \( T' \leftarrow T' \cup \{OBS\} \)
8: end if
9: end for
10: for \( OBS \in T' \) do \( \triangleright \) Stage 2
11: \( (DS, CS) \leftarrow diag(OBS, SD, l) \)
12: \( CS' \leftarrow CS' \cup CS \)
13: end for
14: \( RES \leftarrow SMHS(CS', l) \) \( \triangleright \) Stage 3
15: end procedure
```

As we illustrated in Algorithm 1 our approach is based on the following three stages:

In Stage 1 (Lines 4 to 9), we execute all test cases \( t = (\delta, p) \) (see Def. 1) in our combinatorial test suite \( T \) and for each execution we retrieve a tuple \((v, OBS)\) where \( v \) contains the verdict (either passed or failed) and \( OBS \) is the set of observations obtained for the execution (and thus shall contain also the input stimulus \( \delta \)). Only if a test execution failed, we add the corresponding observations to \( T' \) in Line 7. Thus, upon completion of Stage 1, the set \( T' \) contains for each failed test case execution (and only for those) the corresponding set of observations.

In Stage 2 (Lines 10 to 13), we diagnose in Line 11 all failed test case executions via calling a conflict-driven MBD algorithm \( diag(OBS, SD, l) \) for the diagnosis problem defined by the respective observations, the system description and the cardinality limit \( l \). Candidates for such an algorithm are, e.g., RC-Tree 16 or HS-DAG 15. Please note that the algorithm shall not only return the set of diagnoses \( DS \), but
also the set CS containing all minimal conflicts derived during the computation of DS. In Line 12 we add all conflicts in CS to CS', so that upon completion of Stage 2, CS' contains all conflicts (and only those) that were used to compute the diagnoses for the individual cardinality-restricted diagnosis problems (one problem for each OBS ∈ T').

In the final Stage 3, we compute the minimal hitting sets for CS' with a corresponding cardinality-aware minimal hitting set algorithm MHS(CS, l). Of course, we can again use RC-Tree or HS-DAG for implementing this function, but since we do not have to compute conflicts on-the-fly, we can also use algorithms like the Boolean algorithm [19] that show superior performance but can be used only if the conflicts are known in advance (like it is the case for Stage 3). The result RES contains diagnoses as the MHS of CS', characterizing the problems in S as unveiled by T.

**Corollary 2.** For some diagnosis ∆ ∈ DS for a cardinality-restricted diagnosis problem (OBS, SD, l) considered in Stage 2, RES might neither contain ∆ nor some superset ∆' s.t. ∆ ⊆ ∆'.

*Proof.* (sketch) While one might assume this to be true, especially in the light of how the DAG or tree is maintained for algorithms like [15] or [16], we have to take into account why such constructions need a pruning step. That is, they have to check their results if they find that they used a non-minimal conflict and update their results to how they would have looked like if they had used the newly recognized conflict in the first place. Indeed we require for stage 2 that a solver returns minimal conflicts, which is a requirement many solvers can adhere to. The problem is that it can still happen that some C* = {h1, h2, h47} is a minimal conflict for (OBS, SD, l), and C* = {h1, h3} ⊂ C* is a minimal conflict for some (OBS_2, SD, l). Now assume that no proper subset of C* is contained in CS'. Then those diagnoses (minimal hitting sets) that resolve C* via h47 for (OBS_2, SD, l) might not live up to the requirement described in this corollary, since for the MHS of CS' it suffices to hit and resolve C* (C* is not a minimal conflict in CS').

This corollary shows that with our concept, RES is not simply the set of possible combinations of solutions to the individual problems, but the construction implements the subset-minimality aspect of the definition of a diagnosis. That is, for the example of above any solution that would have resolved C* by h47 would have to contain either h1 or h3 for hitting (resolving) the conflict C*—which obviously would result in a non-minimal solution since either of the two would also resolve conflict C*.

**Theorem 1.** A minimal hitting set ∆ ∈ RES defines an assignment to the health state variables such that for all OBS ∈ T' we have that those observations are consistent with SD under the health state assumptions such that OBS ∪ SD ∪ {h1|h1 ∈ H \ ∆} is satisfiable.

*Proof.* (sketch) We prove this by showing that for every ∆ ∈ RES, and for every OBS ∈ T' there is some ∆' ⊆ ∆ that is a diagnosis for (OBS, SD, l). Since any superset ∆' of a diagnosis ∆ (like ∆) will result in consistency/satisfiability (let us remind you that then even less clauses have to be satisfied by the signals and variables rather than the health state variables), soundness directly follows. Now let us choose some arbitrary OBS ∈ T'. In Stage 2, we computed CS containing the conflicts that allow us to characterize the diagnoses to (OBS, SD, l) as minimal hitting sets of CS. Since ∆ ∈ RES is a minimal hitting set for CS' that in turn is a superset of CS, there exists a minimal subset ∆' of ∆ that hits exactly those conflicts that are in CS' and CS, so that ∆' is a diagnosis for (OBS, SD, l).

The following corollary consequently follows from our proof of Theorem 1:

**Corollary 3.** For every ∆ ∈ RES, and for every OBS ∈ T' there is some ∆' ⊆ ∆ that is a diagnosis for (OBS, SD, l).

Since the characterization of a diagnosis as of Definition 4 does not match a multi-observation setting like ours, let us define the term multi-observation diagnosis in order to characterize ∆ ∈ RES based on Corollary 3.

**Definition 7.** [multi-observation diagnosis] A multi-observation diagnosis for some test suite T and a system description SD is a subset ∆ ⊆ H of the health state variables in SD such that:

- no proper subset of ∆ is also a multi-observation diagnosis
- for every failed test case execution, there is some ∆' ⊆ ∆ that is a diagnosis for that failed test case execution.

**Theorem 2** (soundness). Every ∆ ∈ RES computed by Algorithm 7 is a multi-observation diagnosis.

*Proof.* (sketch) The first requirement that RES shall not contain some ∆' ⊆ ∆ is ensured by the construction of RES, such that all ∆ ∈ RES are minimal hitting sets. The satisfaction of the second requirement follows by Theorem 1 and Corollary 3. The correctness of this theorem then follows directly from the satisfaction of both requirements.

As with any algorithm, as soon as we know that an algorithm is sound (a reported solution is indeed a solution to the problem), we certainly would also like to know whether it is complete such that it would compute all solutions. Since also combinatorial testing is incomplete in the sense that we’re not exercising all possible behavior, we indeed can not conclude that COMBDIAG as depicted in Algorithm 1 computes all possible explanations for all possible behavioral patterns. What we can show though, is that for the observations we have the algorithm is exhaustive, i.e., it will compute all explanations viable for this set of observations satisfying the given cardinality limit l.

**Theorem 3** (exhaustiveness). If there is a multi-observation diagnosis ∆ for some test suite T and a system description SD such that |∆| ≤ l for some given cardinality limit l, then it will be contained in RES as computed by Algorithm 1.

*Proof.* (sketch) By Definition 7 a multi-observation diagnosis ∆ defines a diagnosis ∆' ⊆ ∆ for every test case l ∈ T whose execution failed (s.t. in our Algorithm the corresponding observations are added to T' in Line 7). Such a diagnosis ∆' is an MHS of the minimal conflicts for this observation, where it is important to note that we compute in Stage 2 (Line 11) of our algorithm diagnoses (and the corresponding conflicts to characterize them) up to the same cardinality limit l that serves as limit also for computing RES. This means in turn that all those conflicts necessary to define diagnoses up to size l for these specific observations are added to CS' (Line 12), and we do this for all failed test cases (see Lines 4 to 10).
Now let us assume that there is indeed some $\Delta^*$ that satisfies Def. 7 (and thus is smaller or equal to $\ell$ in size), but is not in $RES$. As outlined at the beginning such a $\Delta^*$ would per definition have to define diagnoses $\Delta^* \subseteq \Delta^*$ for all the failed test cases. Furthermore those diagnoses $\Delta^*$ would have to be smaller or equal in size to $\ell$, and thus would have to be MHS of the conflicts we added to $CS'$ for these test cases in Stage 2, and which are considered by the MHS construction for deriving $RES$ in Stage 3. Thus, by construction, we have that if the MHS and diag algorithms used in Stages 2 and 3 are exhaustive (like the proposed HS-DAG, RC-tree, and Boolean algorithms) are) this would result in a contradiction, so that there indeed can’t be a $\Delta^*$ that satisfies Definition 7 but is not contained in $RES$.

Considering Theorems 2 and 3, we can conclude that our algorithm COMBDIAG is both sound and exhaustive, but it is incomplete in the sense that it only works with the data it has available—which means with an incomplete coverage of the system’s behavior in the testing stage (Stage 1).

5 Some first experiments

Let us now have a look at how our idea works out for our working example and three fault scenarios. To this end, we show in the following table all corresponding signal values.

In principle total coverage of the input behavior would be possible, since $|I| = 3$ so that this would require $2^3 = 8$ test cases. But let us assume that we’re aiming for a combinatorial strength of $s = 2$, which also means that we can re-use the test suites defined in Section 2 for $I = \{a, b, c\}$ and $s = 2$. In the following table, we not only show the input signals for these tests, but also the internal signals $o_1$ and $o_2$, and the single observable system output signal $o_3$ ($O = \{o_3\}$). We do this for the case that the circuit works correctly, and for three scenarios where we inject faults into the circuit. That is, for fault scenario $F_1$, we assume that Gate 1 (the AND gate with $h_1$) behaves like an OR gate, for fault scenario $F_2$ $g_1$ (the AND gate with $h_2$) behaves like an OR gate, and for $F_3$, $g_2$ behaves like an XOR gate.

<table>
<thead>
<tr>
<th>inputs</th>
<th>outputs</th>
<th>nominal</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$ $b$ $c$</td>
<td>$o_1$ $o_2$ $o_3$</td>
<td>$o_1$ $o_2$ $o_3$</td>
<td>$o_1$ $o_2$ $o_3$</td>
<td>$o_1$ $o_2$ $o_3$</td>
<td>$o_1$ $o_2$ $o_3$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The signals for our circuit and various scenarios.

In order to highlight the important parts in this table, we use a grey cell background for all the observable output values if the circuit works correctly and thus exhibits the nominal behavior. These values serve as benchmark for judging whether a test case fails for some fault scenario. For the fault scenarios $F_1$ to $F_3$ we use a grey cell background for the observable output if and only if the test case would fail for this scenario such that the value $o_3$ would be different than for the nominal behavior. This means that for $F_1$ we have that test cases $t_2 \in T$ and $t_3 \in T$ as well as $t_7 \in T'$ would fail. For $F_2$, the failing test cases would be $t_2 \in T$ and $t_3 \in T$ as well as $t_5 \in T'$ and $t_7 \in T'$. For $F_3$, $t_2 \in T$, $t_3 \in T$, and $t_8 \in T'$ would be the failing test cases in our two test suites. Since $T$ and $T'$ are both test suites with $s = 2$, let us now investigate the respective results for our approach when deploying either test suite. For the nominal behavior we can easily see that if the circuit behaves as expected, then there are no failing test cases. Consequently we can observe that $RES$ stays empty for $T$ and $T'$, like we expected since there shall be no multi-observation diagnoses (other than the empty set) in this case.

Now let us investigate fault scenario $F_1$. There, we have that for $T_{t_2}$ as well as $t_{t_3}$ fail. For both of these two test cases, we derive in Stage 2 the diagnoses $h_1$ and $h_3$ and a single minimal conflict $\{h_1, h_3\}$ respectively. The same intermediate results are computed for $T'$ and its failing test case $t_7$. Thus for both test suites we have that when entering Stage 3, $CS'$ contains the single conflict $\{h_1, h_3\}$ and we would compute the two multi-observation diagnoses $h_1$ and $h_3$. Consequently we can observe that for both test suites, the actually injected fault is present as single fault multi-observation diagnosis in $RES$. Obviously there is furthermore no smaller multi-observation diagnosis present in $RES$, but only another one of the same cardinality.

For fault scenario $F_2$, we have that the results for $T$ match those for $F_1$. This stems from the fact that the same test cases fail, and they do so with the same output value so that the observations are the same. Only for $T'$ there is a difference, i.e., that $t_5$ fails in addition to $t_7$ (the latter with the same output value as for $F_1$ though). Since we derive the same diagnoses and minimal conflict for $t_5$ like for $t_7$, the final results for $RES$ stay thus unchanged for both $T$ and $T'$ when compared to $F_1$. The only difference being in the evaluation of the result, since for $F_2$ out of the two candidates offered by both $T$ and $T'$, now $h_3$ catches the fault we injected at Gate 3—which is in contrast to $F_1$, since there $h_1$ for the faulty Gate 1 caught the injected fault.

Fault scenario $F_3$ is different in the sense that we have two conflicts in $CS'$ at Stage 3. For $T$ we have that $t_2$, $t_3$, and $t_4$ fail. For the former two, we have that we get diagnoses $h_1$ and $h_3$ as well as the minimal conflict $C = \{h_1, h_3\}$ as the results of Line 11 for both test cases (again we can see that they fail with the same value for $o_3$ like in the context of $F_1$, so that the computation triggered in Line 11 is in principle the same as for $F_1$). For $t_4$, we compute a minimal conflict $C' = \{h_1, h_2, h_3\}$ with the corresponding diagnoses though. For the computation of $RES$ in Stage 3, $C'$ as derived for $t_4$ will not cause any lasting effect in the MHS computation, since $C'$ is obviously a superset of $C$ that we computed for either of the other two failed test cases. Like we surmised in Section 3, $C'$ is a superset of $C'$ so that it might be computed as minimal conflict for another observation as is illustrated by this example. Consequently, the multi-observation diagnoses in $RES$ are again $h_1$ and $h_3$, catching the fault we injected at Gate 3. Now let us have a look at $T'$. There we have that $t_5$, $t_7$, and $t_8$ fail. For $t_5$ and $t_7$ we compute the minimal conflict $\{h_1, h_3\}$ in Stage 2, and for $t_8$ the diagnoses and the single minimal conflict returned in Line 11 of our algorithm are $DS = \{h_1, h_2, h_3\}$ and $CS = \{\{h_1, h_2, h_3\}\}$ respectively (the same results as
for \( t_3 \). So again we have that the conflict added to \( CS \) for \( t_6 \) in Line 12 will not have any effect since it is a non-minimal conflict in \( CS \) (but a minimal conflict in the local \( CS \)). Thus, also with \( T' \) we get the same multi-observation diagnoses \( h_1 \) and \( h_3 \), with the latter hinting at the actual fault.

Considering these results, we can see that we were indeed always able to find the injected fault with our test suites with a combinatorial strength \( s < |I| \), regardless of the fault scenario, and which of the two combinatorial test suites were used. We also saw for the third fault scenario, that indeed not all test cases provide the observations to rule out Gate 2 as source of the problem, but that computing multi-observation diagnoses using a combinatorial test suite allowed us avoid such a situation.

Please note that we considered a limited observability of the circuit’s signals on purpose, since if \( o_1 \) is observable as well, this would only make the diagnostic process more succinct with less diagnoses and conflicts having to be handled by our algorithm. That is, if we can observe \( o_1 \) this would mean for all of our fault scenarios \( F_1,F_2, \) and \( F_3 \), that the injected fault indeed results in a single fault conflict for any of the failed test cases respectively (\( h_1 \) for \( F_1 \) and \( h_3 \) for \( F_2 \) and \( F_3 \)). Being able to observe \( o_2 \) would not have an effect for the fault scenarios considered though. Thus, we chose a limited observability scenario for a more elaborate illustration of our concept.

6 Related work

From an abstract point of view, sequential diagnosis [20], where we focus on determining new “measurements” to retrieve new data for being able to discriminate between diagnoses is somehow related to our research. While also there the aim is to enrich the data used in the reasoning process, the focus there is on determining which data queries would allow to discriminate between diagnoses. In our case, it is pre-defined though which data we can use, and we do not construct any distinguishing tests [8] like they might be used also by robots to decide between hypotheses about the current system state. If one had the option to tune the generation of the combinatorial test suite though, research in the context of sequential diagnosis might be of interest.

For such optimizations, we consider research regarding fault diagnosability in the area of discrete event systems [21] to be related even closer, since it might allow us to assess corresponding constraints and restrictions. Compared to our approach presented in this manuscript, i.e., where we have a predefined combinatorial test suite, again we can see that the reasoning behind such work has a different focus though. For future experiments that aim at determining and assessing reasonable values for the combinatorial strength parameter and corresponding fault identification guarantees, or in other words the (possibly guaranteed) quality of the process, research regarding fault diagnosability could certainly provide interesting research directions for such extensions.

Our concept as presented in this manuscript is not the first that was aimed at combining the techniques of combinatorial testing and diagnostic reasoning. In [18] and [6], for instance, we explored options to combine research from both areas in order to derive a knowledge-base that is extensive enough to support an abductive diagnosis process. Using also fault injection, the aim there was to define the parameters for a set of simulations that would allow us to observe and afterwards encode knowledge about how faults affect a system’s behavior in the desired knowledge base. Consequently and in contrast to our current work, also the fault combinations (the health state variables) were part of the input model for the MCA generation step for those papers, and we focused there on designing and optimizing this generation process for the desired abductive diagnosis knowledge-base by exploiting the combinatorial exploration concept in the context of diagnostic reasoning. Our current work is thus significantly different, since we’re relying on combinatorial test suites and connect them with consistency-oriented model-based diagnosis in order to extract from a respective test suite’s execution data the diagnostic information about which components could explain the failed test cases if we assume them to behave in a faulty way.

7 Conclusions

Assessing the quality of a system is certainly a challenging task, and any help we can get in this process is desirable. Testing is still a key concept deployed in this context, since it scales from models to real hardware, and aside its flexibility as well as the option to be able to consider the corresponding test execution data whenever executing an individual test case finished (for model-checking, for instance, we have to wait until the algorithm finishes the entire computation), it is a concept as intuitive as it can get: we examine whether the system behaves as expected for some exemplary behavioral stimuli.

Diagnosis as a technique that addresses the task of identifying those components or parts of a system that can explain what went wrong if we encounter some problem is certainly a powerful tool as well. Despite its obvious benefits, it is not a technique present in every development tool chain yet. In this paper, we showed that we can easily combine consistency-oriented model-based diagnosis with combinatorial testing though, and we highlighted the advantages of combining the two techniques for our approach.

That is, with combinatorial testing we can structurally cover the behavioral space of a system, in order to derive a test suite that considers all possible variable interactions of a certain size. Meaning that for any variable subset up to this size, all possible variable assignments are exercised by at least one test case. In principle this could be scaled also to a totally exhaustive behavioral coverage, but since this is infeasible in practice for most systems, we’re focusing on some desired combinatorial strength instead, and related work [13] showed that there are reasonable suggestions for choosing a value for this parameter in a testing context. A main task after executing any such test suite is the evaluation of the execution data and in turn identifying the causes of any failed test cases. This is where model-based diagnosis can step in. For our approach, we use a consistency-based MBD solution in this context and showed in this paper that we can indeed successfully combine these two techniques for a mutual benefit. That is, we need observations for deploying MBD, and when testing a system we need a technique pinpointing us to problems in the system via analyzing test execution data.

Via defining multi-observation diagnoses, we established a corresponding formal framework for being able to effectively aggregate data from the monotonic diagnostic reasoning for individual failed test cases in a non-monotonic approach that takes observations for multiple and different stimuli into account. We showed that at least for our exper-
iments, the combinatorial test suite provided enough data to avoid the minor quality of the data retrieved for individual test cases (e.g. in the context of $F_3$ for our example circuit). By focusing on the conflicts computed for individual failed test cases in our non-monotonic global approach, and defining specific limits we could show that we can efficiently and effectively compute the desired multi-observation diagnoses with an algorithm that we can scale in respect of some desired cardinality limit. Based on all of our algorithm’s parameters, we furthermore discussed (and proved) soundness and the exhaustive nature of our algorithm, as well as discussed completeness in the context of the limitations of combinatorial testing. 

While we showed in our discussion and first experiments that the concept works out as expected in principle, future work will have to investigate the practical applicability and scalability of our concept in the context of real-world problems from a variety of domains. That is, while related research suggests a desirable combinatorial strength, future experiments will have to investigate the effects of this parameter on the diagnostic process and the achieved quality of the diagnoses. It will also be of interest to use the proposed algorithm as benchmark for the abductive reasoning suggested in [18][6] and evaluate as well as compare achievable results.

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References