Integrating multiple observation sets into consistency-based diagnosis

Franz Wotawa[1]
[1]Institute for Software Technology, TU Graz
e-mail: wotawa@ist.tugraz.at

Abstract

Consistency-based diagnosis utilizes models of the system together with observations for computing diagnosis candidates. Distinguishing these candidates – if even possible – requires obtaining new observations. In model-based diagnosis procedures for selecting the next best measurement have been described and evaluated. In this paper, we follow an alternative research direction making use of different set of observations of a system, and how they can be integrated into the consistency-based diagnosis framework. Besides extending basic definitions, we show how to represent the diagnosis problem utilizing formalisms of answer set programming. In addition, we show how spectrum-based fault localization can be integrated within the framework to allow utilizing observations that do not lead to conflicts.

1 Introduction

Diagnosis, i.e., detecting failures, localizing faults, and repair, is an important task in our daily live ranging from maintaining engineered systems to the medical domain. Since the beginnings of Artificial Intelligence, the automation of diagnosis have been of interest leading to model-based diagnosis [1; 2], which come in two flavors, i.e., consistency-based diagnosis [3; 4] and abductive diagnosis [5]. Whereas consistency-based diagnosis relies on models of the correct behavior of system components, abductive diagnosis utilizes knowledge about how systems behave in case of faults to compute diagnosis candidates for identifying root causes. There is work on integrating fault models into consistency-based diagnosis (see e.g. [6]) and how to integrate knowledge about the correct behavior into abductive reasoning (see for example [7]).

Although, originally model-based diagnosis mainly has focused on fault detection and localization there has been work on repair, e.g., [8; 9] and more recently [10], as well. In addition, research include work on improving diagnostic reasoning using physical impossibilities [11], distinguishing diagnosis candidates utilizing additional measurements [4], and diagnosis algorithms (see [12] for a comparison of the runtime of different diagnosis algorithms). However, there has been only little work on integrating multiple observation sets. In most cases, the diagnosis problem comprises a diagnosis model, i.e., a formal description of the structure and the behavior of components, and one set of observations.

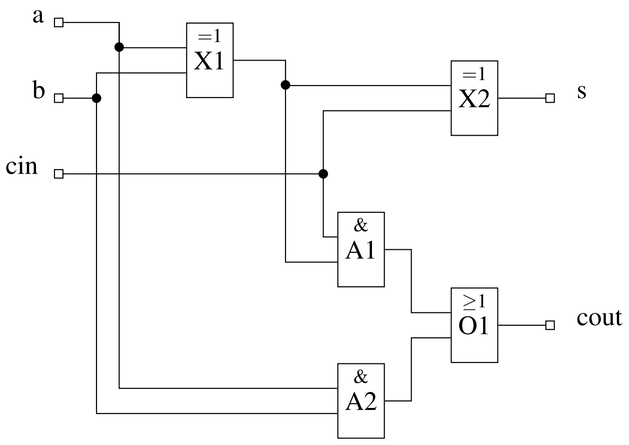

Figure 1: A digital full adder circuit.

Table 1: Expected behavior and observed behavior of a full adder considering a stuck at 0 fault of component O1. Faulty outputs are marked with *.

	Input			*Exp. Behav.*		*Obs.*	
#	a	b	cin	s	cout	s	cout
1	0	0	0	0	0	0	0
2	0	0	1	1	0	1	0
3	0	1	0	1	0	1	0
4	0	1	1	0	1	0	0*
5	1	0	0	1	0	1	0
6	1	0	1	0	1	0	0*
7	1	1	0	0	1	0	0*
8	1	1	1	1	1	1	0*

This is an interesting observation because in other areas like software debugging a lot of work have been published dealing with identifying root causes of failures considering the program executions for several test cases. Spectrum-based fault localization [13; 14] extracts likelihood measures for each statement of being correct or faulty from several executions taking care of knowledge regarding whether an execution leads to a failure or not. Accordingly to [15], spectrum-based fault localization can be considered as the one having the largest share on publications in the area of automated and algorithmic software debugging.

In this paper, we contribute to the use of multiple observation sets for diagnostic reasoning focusing on consistency-based diagnosis. In particular, we are interested in showing how multiple observation sets can be integrated into the foundations behind consistency-based diagnosis as well as how to compute diagnosis. The work, presented in this paper, is similar to [16] but in contrast does not utilize hitting set computation for obtaining diagnoses. Instead, we make use of answer set programming [17] for diagnosis and follow previous work (e.g., see [18; 19]). Other recent work, dealing with the integration of multiple observations include [20; 21]. The authors of [20] extended the definitions of diagnosis to handle multiple observations whereas we provide one general definition. In [21], the authors distinguish different kinds of models handling strong and weak fault models, and distinguishing intermittent and non-intermittent faults. In this paper, we solely focus on consistency-based diagnosis, i.e., considering weak fault models.

Let us describe the multiple observations diagnosis problem using a small example. In Figure 1, we depict the schematics of a digital full adder comprising two exclusive-or gates X1 and X2, two and-gates A1 and A2, and one or-gate O1. In Table 1 we see the expected as well as the observed behavior of the full adder assuming a stuck at 0 fault of O1. In practice, we may not observe all the different input-output as given in the table but only a subset of these, e.g., line 1, 2, 4, 6, 7, 8. The question is how to combine these observations in order to come up with a single diagnosis for the full adder and all given observation sets? One way of dealing with this problem is to compute all hitting sets for each observation set and to make use of them to come up with a final hitting set graph like outlined in [16]. In this case, all conflicts contribute to the computation of diagnosis. In this paper, we follow a different solution where we compute diagnoses directly from the model without the need for computing conflicts. The open question is, how to integrate multiple observation set in such a setup?

In addition, we may be interested in answering how to integrate observations not leading to contradictions, into diagnostic reasoning. In the classical setup of consistency-based diagnosis (and discussed in detail in this paper as well), such observations do not contribute to diagnosis in general. Hence, we have to discuss whether there are certain assumptions under which we are able to integrate such observations, or alternative approaches for utilizing information from non-conflicting observations.

We organize the paper as follows: First, we discuss the basic foundations when dealing with multiple observation sets. We further show how such diagnosis problems can be represented as answer set programs. Afterwards, we discuss the computation of diagnoses re-examining appropriate algorithms. Furthermore, we investigate the problem of integrating observation sets not leading to contradictions. Finally, we conclude the paper and outline open research questions.

2 Foundations

In the following, we briefly outline the underlying formal definitions behind consistency-based diagnosis following Reiter [3]. We slightly adapt the definition of diagnosis to handle multiple observation sets. We first introduce the formal representation of a system, i.e., a diagnosis system.

Definition 1 (Diagnosis system)**.** *A* diagnosis system *is a tuple* $(SD, COMP)$ *where SD is a formal description of the system to be diagnosed, i.e., the model, capturing the correct behavior of components, and COMP is the set of components.*

It is worth noting that the model *SD* formalizes the correct behavior of a component $c \in COMP$ only. For this purpose, we introduce a predicate *ab* with a component as the only parameter, stating that the component is abnormal, i.e., not working as expected. In *SD*, we state the behavior of *c* using an implication, i.e., $\neg ab(c) \rightarrow (Behav)$, where *Behav* formalize the behavior of *c*. See [6] for more details on modeling and underlying principles (including how to handle fault models, which we are ignoring in this paper). Using diagnosis systems together with a set of observation sets, we are able to specify a diagnosis for all given different observation sets.

Definition 2 (Diagnosis)**.** *Given a diagnosis system* $(SD, COMP)$ *a set* $\{OBS_1, \ldots, OBS_k\}$ *where each* OBS_i *(for* $i = 1 \ldots k$*) is a distinct set of observations. A set* $\Delta \subseteq COMP$ *is a* diagnosis*, if and only if for all* $i \in \{1, \ldots, k\}$ *the logic sentence* $SD \cup OBS_i \cup \{ab(C) | C \in \Delta\} \cup \{\neg ab(C) | C \in COMP \setminus \Delta\}$ *is consistent.*

In Definition 2 we do not make any assumptions regarding the observation sets OBS_i. They may specify different values for connections, signals, or any other observable entity. Two observation sets may specify different output observations for the same inputs of a system as well. Such a case may exist in case of intermittent faults. In contrast to [21], we do not specifically focus on differentiating intermittent behavior from non-intermittent one. The proposed approach is still able to deliver diagnoses in this case. However, it would be of interest to investigate on the effect of intermittent behavior to the diagnosis results.

In practice, we are interested not in all diagnoses but those having specific properties, e.g., being minimal. This leads to the definition of parsimonious diagnoses.

Definition 3 (Parsimonious diagnosis)**.** *A diagnosis* Δ *of a diagnosis systems is said to be* parsimonious *(or* minimal*) if and only if there is not* $\Delta' \subset \Delta$ *that is itself a diagnosis accordingly to Definition 2.*

Note that Definition 2 directly extends the original definition of diagnosis from [3] to be able handling multiple observations. This is different from other previous work, i.e., [20], where the authors come up with additional definitions for the same purpose. As a consequence, we do not need to introduce different concepts for handling minimality, i.e., minimal diagnosis (for the case of a single observation) and redundant diagnosis (when multiple observations are present). Furthermore, we do not need to combine all diagnosis from the observations and rule out redundant diagnoses afterwards. Instead we are able to compute parsimonious diagnoses for all observations at once.

It is worth noting that in practice we focus not only on obtaining parsimonious diagnosis but also limit diagnosis size, i.e., $|\Delta|$ to pre-defined values, e.g., 1, 2 or 3. The motivation behind this restriction is that a higher cardinality diagnosis is less likely to occur. Moreover, parsimonious diagnoses characterize all diagnoses in consistency-based diagnosis. The following theorem states this:

Theorem 1. *Given a diagnosis system* $(SD, COMP$ *and a set of observation sets* $\{OBS_1, \ldots, OBS_k\}$*. Any diagnosis is either a parsimonious diagnosis or a superset of a parsimonious diagnosis.*

Theorem 1 states that all diagnoses are supersets of parsimonious diagnoses. Hence, the set of all parsimonious diagnoses characterizes all diagnoses for a given system and set of observation sets. The theorem follows almost directly from Definition 2 and the way the correct behavior stated in *SD* is formalized. If we set more components to *ab*, we can only derive the same or less predicates. Hence, we are not able to come up with a conflict in this case and a superset of a parsimonious diagnosis must be a diagnosis too.

Note that there is the underlying assumption behind Definition 2 that the system from which we obtain the different sets of observations does not change. Hence, there is neither an internal nor an external mechanism that adapt the system after any observation. Hence, all observation sets obtained correspond to one and only one health state of a system. It is worth noting, that we may have different faults in the system and that two observation sets reveal one or the other fault but not necessarily both. This underlying assumption is in line with the definitions of [20].

3 Modeling for diagnosis

In this section, we discuss modeling for diagnosis using answer set programming [17]. In particular, we make use the input language used by the answer set programming tool `clingo` [22; 23], which relies on Prolog [24] syntax and includes extensions. Hence, we first state the diagnosis model capturing the behavior of components and their interconnections using Prolog syntax before making use of specific ASP features allowing us to compute diagnosis candidates directly using `clingo`. The advantages of using ASP for diagnosis are: (i) the expressivity of the provided language for specifying models in logic, and (ii) the availability of fast solvers allowing to compute diagnoses for even larger systems (see [19]).

In order to capture components, connections, and values, we introduce predicates `comp\1`, `conn\2`, and `val\3` respectively. A component has a unique name and is from a particular type like an and-gate. Hence, we make use of a predicate `type\2` to set the type of a component. Ports of components are stated as functions with the name of the component as parameter. Connections are assumed to connect two ports of not necessarily different components. The values are for a port and belong two exactly one observation set.

Let us first, introduce a model for a digital inverter using the mentioned predicates:

```
val(Tc,out(C),Z) :- type(C,T), val(Tc,
    in(C),Y), tuple(T,Y,Z), nab(C).
val(Tc,in(C),Y) :- type(C,T), val(Tc,
    out(C),Z), tuple(T,Y,Z), nab(C).

tuple(inv,0,1).
tuple(inv,1,0).
```

The first two lines state the behavior of any component `C` of type `T` comprising two ports `in` and `out`. In particular it states that if the component `C` is not abnormal, i.e., `nab(C)` is true, there is a value for the input (output) given, which is determined using the predicate `tuple`, we are able to conclude the value of the output (input) of `C`. These two lines can also be used to state the behavior of a buffer component. We only need to add information regarding the expected input and output pairs like follows:

```
tuple(buff,0,1).
tuple(buff,1,0).
```

Using this definitions, we can specify a particular inverter `I1`, using the predicate `type` as follows: `type(inv,i1)`. Note that we follow Prolog syntax here and formalize constants like `I1` as strings starting with a lowercase letter. What is missing is how to define connections? This can be done in a similar way, stating that connected ports share the same values:

```
val(Tc,X,V) :- conn(X,Y), val(Tc,Y,V).
val(Tc,Y,V) :- conn(X,Y), val(Tc,X,V).

:- val(Tc,X,0), val(Tc,X,1).
```

In the last line, we state that a connection can only take one Boolean value at a time. Hence, the port's value of `X` is maybe true (`1`) or false (`0`) but never both.

Using the model we are now able to define two inverters and connect them:

```
type(inv,i1).
type(inv,i2).
conn(out(i1),in(i2)).
```

Before discussing other digital components, we further define the relationship between components and their types. Every component `X` has a type `T`, which we formalize as follows:

```
comp(X) :- type(X,T).
```

For components with two inputs `in1`, `in2`, and one output `out`, we come up with similar Prolog rules for stating the behavior. However, there are differences because in the backward direction (i.e., from the output to the inputs), we sometimes cannot uniquely come up with a value. Hence, we have to find such scenarios and represent them in a way avoiding different values to be assigned to a port, which would lead to inconsistencies.

```
val(Tc,out(C),Z) :- type(C,T), val(Tc,
    in1(C),V), val(Tc,in2(C),W), tuple(T
    ,V,W,Z), nab(C).

val(Tc,in1(C),1) :- type(C,and), val(Tc
    ,out(C),1), nab(C).
val(Tc,in2(C),1) :- type(C,and), val(Tc
    ,out(C),1), nab(C).
val(Tc,in1(C),0) :- type(C,and), val(Tc
    ,out(C),0), val(Tc,in2(C),1), nab(C)
    .
val(Tc,in2(C),0) :- type(C,and), val(Tc
    ,out(C),0), val(Tc,in1(C),1), nab(C)
    .

val(Tc,in1(C),0) :- type(C,or), val(Tc,
    out(C),0), nab(C).
val(Tc,in2(C),0) :- type(C,or), val(Tc,
    out(C),0), nab(C).
val(Tc,in1(C),1) :- type(C,or), val(Tc,
    out(C),1), val(Tc,in2(C),0), nab(C).
val(Tc,in2(C),1) :- type(C,or), val(Tc,
    out(C),1), val(Tc,in1(C),0), nab(C).
```

```
val(Tc,in1(C),V) :- type(C,xor), val(Tc
    ,out(C),Z), val(Tc,in2(C),W), tuple(
    xor,V,W,Z), nab(C).
val(Tc,in2(C),W) :- type(C,xor), val(Tc
    ,out(C),Z), val(Tc,in1(C),V), tuple(
    xor,V,W,Z), nab(C).

tuple(and,0,0,0).
tuple(and,1,0,0).
tuple(and,0,1,0).
tuple(and,1,1,1).

tuple(or,0,0,0).
tuple(or,1,0,1).
tuple(or,0,1,1).
tuple(or,1,1,1).

tuple(xor,0,0,0).
tuple(xor,1,0,1).
tuple(xor,0,1,1).
tuple(xor,1,1,0).
```

The first line represents the case of reasoning from the inputs to the output that generally holds for all components regardless of their type (assuming that in this direction, we always implement a function using the `tuple` predicate). The other lines are specifically for the different component types `and`, `or`, and `xor`. Similarly, other digital gates like nand, nor, or xnor, can be formalized using Prolog syntax.

Using the described model of digital components and their connections, we can easily represent the full adder depicted in Figure 1 in Prolog:

```
type(x1,xor).
type(x2,xor).
type(a1,and).
type(a2,and).
type(o1,or).

conn(a,in1(x1)).
conn(b,in2(x1)).
conn(out(x1),in1(x2)).
conn(cin,in2(x2)).
conn(out(x2),sum).
conn(cin,in1(a1)).
conn(out(x1),in2(a1)).
conn(a,in1(a2)).
conn(b,in2(a2)).
conn(out(a2),in2(o1)).
conn(out(a1),in1(o1)).
conn(out(o1),cout).
```

In this model, we also have constants for the inputs `a`, `b`, `cin`, `s`, and `cout` for which we are specifying the observations. Let us assume that we have observations for lines 1, 2, 4, 6, 7, 8 of the tuples given in Table 1. The observation sets o_i of any line i can be represented using the `val` predicate as follows:

```
val(o1,a,0).
val(o1,b,0).
val(o1,cin,0).
val(o1,sum,0).
val(o1,cout,0).
....
val(o8,a,1).
val(o8,b,1).
val(o8,cin,1).
val(o8,sum,1).
val(o8,cout,0).
```

The described model together with the observation can be used for diagnosis and any theorem prover that is able to check consistency of Prolog programs accordingly to Definition 2. We only need to set `nab` predicates of components to true, i.e., adding them as fact, that are assume to work correctly. However, we can use `clingo` directly for computing diagnosis of a given size as being discussed in the next section.

4 Computing diagnoses

Computing diagnosis using answer set programming and in particular `clingo` has already been described previously [19]. To be self-contained we briefly recapitulate how to encode diagnosis using `clingo`. Given a propositional theory (i.e., a logic program), an answer set is a satisfiable set of propositions that can be derived from the answer set program in an acyclic way. For example, the `clingo` program:

```
a :- b.
b.
```

has the one answer set `a`, `b` because both propositions can be derived from the program. However, the program

```
a :- b.
b :- a.
```

has an empty answer set, because neither `a` nor `b` can be derived in an acyclic way. Let us consider now the following program:

```
a :- not b.
b :- not a.
```

When assuming `a` (`b`) to be true, its negation is false, and `b` (`a`) cannot be derived. Hence, we have two answer sets for this program, i.e., `a` and the other one only comprising `b`. This example, can be directly used for diagnosis because it enables the answer set program solver to decide either `a` or `b`. In case of diagnosis, we come up with the following two rules establishing the relationship between the predicates `ab` and `nab` defined for a certain component:

```
nab(X) :- comp(X), not ab(X).
ab(X) :- comp(X), not nab(X).
```

When adding these rules to the model, `clingo` can decide on the health state of components to make the program comprising the logic model together with the given observations consistent. Hence, all obtained answer sets are diagnoses. In order to improve this process, we further want to formalize that we are interested of all diagnoses of a particular pre-defined size. For this we use the extensions `clingo` offers.

```
numABs(N) :- N = #count { C : ab(C) }.
:- not numABs(Ds).
```

Algorithm 1 ASPDiag(M,n)

Input: An ASP diagnosis model M, and the desired cardinality n
Output: All minimal diagnoses up to n

```
 1: Let $DS$ be $\{\}$
 2: Let $M_f$ be $M$.
 3: for $i = 0$ to $n$ do
 4:     $M'_f = M_f \cup \{$ :- not numABs($i$). $\}$
 5:     $S = \mathscr{F}\left(\textbf{ASPSolver}(M'_f)\right)$
 6:     if $i$ is 0 and $S$ is $\{\{\}\}$ then
 7:         return $S$
 8:     end if
 9:     Let $DS$ be $DS \cup S$.
10:     for $\Delta$ in $S$ do
11:         Let $C = AB(\Delta)$ be the set $\{c_1, \ldots, c_i\}$
12:         $M_f = M_f \cup \{$ :- ab($c_1$), ..., ab($c_i$). $\}$.
13:     end for
14: end for
15: return $DS$
```

In the first line, we count the number of abnormal predicates obtained when search for an answer set, and store the result in predicate `numABs\1`. In the second line, we state that we are only interested in diagnoses of size `Ds`. During computation of diagnosis, we set `Ds` to one value starting from 0 to the maximum boundary. Note that when setting `Ds` to 1, `clingo` returns single fault diagnosis. It is also worth noting that for a general diagnosis algorithm, we have to further assure that no supersets of diagnoses are generated when increasing the value of `Ds`. This can be assured by adding constraints stating that certain `ab` predicates should not be set to true.

Algorithm 1 ASPDiag (taken from [19]) implements the computation of diagnoses described before, making use of `clingo` as the **ASPSolver** solver. It is worth noting that the first parameter of the ASPDiag algorithm, i.e., model M, comprises all rules described before except the rule for stating the size of diagnoses to be computed. This information is updated during the execution of ASPDiag.

It is easy to see that ASPDiag terminates computing all parsimonious diagnoses for a given model, which comprises the coding already described in a previous section for the system and the set of observation sets. When using this model together with `clingo` for the full adder, we obtain one answer set `nab(x1), nab(x2), nab(a1), nab(a2), ab(o1)` requiring a neglectable amount of time (less than 1/100 seconds). It is worth noting that if we only use the set of observations `o8`, we receive two answer set: One is the same than before and the other one is `nab(x1), nab(x2), nab(a1), ab(a2), nab(o1)`. Hence, we see that considering multiple observation sets may allow to reduce the number of diagnoses.

When using the full adder model algorithm ASPDiag delivers the following 3 parsimonious diagnoses:

- one single fault diagnosis $\{\text{o1}\}$,
- one double fault diagnosis $\{\text{a1}, \text{a2}\}$, and
- one triple fault diagnosis $\{\text{x1}, \text{x2}, \text{a2}\}$.

In the next section, we discuss properties and limitations of the presented consistency-diagnosis approach integrating multiple observation sets.

5 Discussion

In this section, we discuss properties of the presented foundations, answer the question regarding the influence of particular observation sets to the computed diagnoses, and present a solution for integrating observations not leading to conflicts into the framework. Obviously, corresponding theorems from [3] still hold in the context of this paper. This is due to the fact, that the way of handling multiple observations can be represented as conjunction of system models for each observation set like done in our answer set programming representation. Therefore, we can safely state that a diagnosis exists if and only if $SD \cup OBS_i$ is consistent for all given observation sets $OBS_1, \ldots, OBS_k$.

Furthermore, by construction there is a relationship between diagnoses obtained for each OBS_i and the overall diagnoses for all observation sets, which we summarize in the following theorem:

Theorem 2. *Any parsimonious diagnosis Δ for the diagnosis system $(SD, COMP)$ and the set of observation sets $\{OBS_1, \ldots, OBS_k\}$, is also a diagnosis for $(SD, COMP)$ and the set $\{OBS_i\}$ for $i = 1 \ldots k$, but not vice versa.*

We argue as follows: All diagnoses for any OBS_i must contribute to the overall diagnoses. It might be the case that such a diagnosis is not in the result of diagnosis considering all observations. However, a superset of this diagnosis must be present. This is due to the fact that otherwise, the corresponding overall diagnosis can never be consistent. Because of the fact that every superset of a diagnosis is itself a diagnosis (Theorem 1), we can conclude that every diagnosis for all observation sets must be a diagnosis when considering only one particular observation set. The opposite is not true. For this purpose, we only need to consider the full adder example. There are observations leading to the empty set being a diagnosis. However, the empty set is not a diagnosis for the overall set of observation sets.

Taken the last paragraph into account it seems that not every observation set OBS_i is important to further constraint the computation of diagnoses. OBS_i leading to an empty diagnosis are not used to further restrict the search space, which is the case in other methods like spectrum-based fault localization mentioned in the introduction. Let us partition the set of observation sets into those leading to the empty diagnosis and the others. Similarly in system testing used test cases can be characterized as passing or failing. In the following definition, we adopt this concept and apply it to observation sets.

Definition 4 (Failing/passing observations)**.** *Given a diagnosis system $(SD, COMP)$. An observation OBS is said to be failing (passing) if and only if $SD \cup OBS \cup \{\neg ab(C) | C \in COMP\}$ is inconsistent (consistent)*

From any set of observation sets O we can obtain the set of failing and passing observations as follows:

$$FAIL(O) = \left\{ OBS \middle| OBS \in O \wedge \left(\begin{array}{c} SD \cup OBS \cup \\ \{\neg ab(C) | C \in COMP\} \models \bot \end{array} \right) \right\}$$

$$PASS(O) = O \setminus FAIL(O)$$

We use these definitions of passing and failing observation sets in the next theorem stating that only failing observation sets lead to (non-empty) diagnoses we are interested in.

Theorem 3. *Let D_Δ be the set of all parsimonious diagnoses obtain using the diagnosis system $(SD, COMP)$ and the set $\{OBS_1, \ldots, OBS_k\}$ of observation sets. D_Δ is equivalent to the set of all parsimonious diagnoses obtain for the same diagnosis system when considering failing observations $FAIL(\{OBS_1, \ldots, OBS_k\})$ only.*

The theorem holds because all observation sets in $PASS(.)$ deliver the empty diagnosis where all of its supersets can be diagnoses.

An immediate consequence of Theorem 3 is that we can safely ignore passing observation sets when computing diagnoses and focus only on failing one. This may potentially reduce the time required for diagnosis. On the downside, a direct use of passing observation sets in consistency-based diagnosis is not possible. However, the question remains, whether it is still possible to utilize passing observation sets for reducing the number of diagnosis candidates. We are going to (at least partially) answer this question in the following section.

6 Integrating non-conflicting observations

In order to deal with multiple observation sets, we adopt ideas from spectrum-based fault localization [13; 14] where a suspiciousness value is computed for each statement in a program considering passing and failing runs of a program. Instead of taking care of the semantics of statements, only execution traces are used. The underlying motivation is that statements never executed in failing test cases can be considered as never causing wrong behavior, whereas statements always executed in failing but never in passing runs are more likely to be responsible for detected failures. Regarding statements that are both executed in some passing and some failing runs we cannot classify them as being responsible or not for a faulty behavior. In spectrum-based fault localization a similarity index is computed ranging between 0.0 and 1.0 indicating the degree of responsibility to a failure.

In the context of digital circuits (but also other hardware), we do not have statements that are executed or not executed during computation. For the full adder every run causes the components to compute output values given certain input values. In order to cope with this issue, we make use of the same ideas already used for spreadsheets where we have a similar situation. In [25] the authors present a comparison of 42 similarity coefficients applied to fault localization of spreadsheets. To make use of spectrum-based fault localization, the authors introduced the concepts of cones, i.e., components that lie on a path between a given output, and the input of a given system. Hence, instead of the whole system, we only consider the structure, for fault localization.

Note that the use of cones for spectrum-based fault localization is different from original work, where the spectrum, i.e., the executed and not executed statements for different observations or test cases, is based on the control dependencies of a program. In case of spreadsheets or hardware explicit control is not that often used. Hence, we have to rely on data dependencies that are captured using the concept of cones. It would also be possible to extend the application of spectrum-based fault localization applied to program debugging utilizing cones as well.

To adopt the ideas of [25], we assume that we know for any component $C \in COMP$ all other components where their output is connected with an input of C. This information is accessible using a function ρ, i.e., $\rho(C) = \{C' | C' \in COMP$, and the output of C' is connected with an input of $C\}$. Using ρ we can easily obtain cones recursively:

$$cone(C) = \{C\} \cup \bigcup_{C' \in \rho(C)} cone(C')$$

Note that in the following we assume that each component has only one output, so that we can identify the output of the system by the last component connected with the output. Hence, there is a 1:1 relationship between as system output and one component. For our full adder example, component x2 corresponds to the output s, and o1 to cout. Therefore, we extend the definitions of cones to the outputs as well, e.g.:

$$cone(\mathrm{s}) = cone(\mathrm{x2}) = \{\mathrm{x1}, \mathrm{x2}\}$$
$$cone(\mathrm{cout}) = cone(\mathrm{o1}) = \{\mathrm{x1}, \mathrm{a1}, \mathrm{a2}, \mathrm{o1}\}$$

In addition, to cones we also need to know whether an output (or its corresponding component) is wrong, i.e., contradicting a value stated in a set of observations OBS_i considering, the given input values. Using this information, we come up with a hit spectrum, where we have a column for each output and set of observations, and a row for each component, and a final row where we state whether the output is erroneous or not. If there is an error in the output for an observation, we mark the corresponding entry of the error with • and leave it blank, otherwise. For the other entries, we mark them with • if the corresponding component in the row is element of *cone* of the output. When using this process we obtain the hit spectrum given in Table 2 for our full adder example.

Table 2 not only comprises information about whether a component causes a certain output value, but also n_{ij} values on the right of the table. n_{ij} is used to count the number of • in a row under particular conditions:

n_{11}...	Number of • in the row for faulty output values.
n_{10}...	Number of • in the row for correct output values.
n_{01}...	Number of blanks in the row for faulty output values.
n_{00}...	Number of blanks in the row for correct output values.

These n_{ij} figures basically indicate whether a certain component when being responsible or not, causes an error or not. In spectrum-based fault localization these values are used for computing similarity indices. One often used index, providing the one of the best rankings for components in the case of program debugging is the Ochiai index defined as follows:

$$c_{ochiai} = \frac{n_{11}}{\sqrt{(n_{11} + n_{10}) \cdot (n_{11} + n_{01})}}$$

In Table 2 we find the Ochiai coefficient for all components of the full adder considering the 6 set of observation sets. It is worth noting that for this example, the outcome of the coefficient does not state any difference except for component x1. Depending on the outcome of course these coefficient values vary. Note also that the presented approach

Table 2: Hit spectrum of the full adder together with .

	s						cout						n_{11}	n_{10}	n_{01}	n_{00}	c_{ochiai}
	o1	o2	o4	o6	o7	o8	o1	o2	o4	o6	o7	o8					
x1	●	●	●	●	●	●							0	6	4	2	0.00
x2	●	●	●	●	●	●	●	●	●	●	●	●	4	8	0	0	0.58
a1							●	●	●	●	●	●	4	2	4	2	0.58
a2							●	●	●	●	●	●	4	2	4	2	0.58
o1							●	●	●	●	●	●	4	2	4	2	0.58
error									●	●	●	●					

extends the one described in [25] allowing to specify not only one set of observations but sets of a set of observations.

What is missing is the integration of the similarity indices into ordinary consistency-based diagnosis. In [4] the authors state how to compute the probability of a given diagnosis using the fault probability of the components. For this purpose, we only need to consider the probability that all components in the diagnosis are faulty and all other components are working as expected. When additionally assuming that the health statuses of components are stochastically independent, we obtain the following equation:

$$p(\Delta) = \prod_{C=\Delta} p_F(C) \cdot \prod_{C \in COMP \setminus \Delta} (1 - p_F(C))$$

For combining coefficients into this framework, we argue as follows: First, similarity coefficients have (similar to probabilities) values ranging between 0 and 1. Second, the similarity coefficients indicate suspiciousness of components. Hence, similar to p_F they are intended to be a measure of the degree of being faulty for components. Therefore, we state that they estimate the fault probability of components, i.e., $p_F(C) \sim c(C)$. Note that the fault probabilities obtained using a similarity coefficient reflect the situation once at least one failing observation is available. The fault probabilities usually attached to hardware components are usually based on the mean time to failure and have a more predictive measure.

However, when estimating the fault probabilities using similarity coefficients c directly, we may face anomalies. If the similarity indices are larger than 0.5, larger diagnoses become more likely, which is – in practice – not often the case. To avoid such anomalies but do not change any ranking imposed by the similarity coefficients, we define p_F using c as follows:

$$p_F(C) = 0.1 \cdot c(C)$$

The introduction of the factor 0.1 is a pragmatic choice and more research is required for clarifying limitations and the impact of choices regarding the computation of p_F using c. This requires a sophisticated experimental evaluation.

Using the Ochiai coefficient for our full adder example, we obtain the following probabilities for the three diagnoses:

Diagnosis Δ	$p(\Delta)$
$\{o1\}$	0.04830
$\{a1, a2\}$	0.00296
$\{x1, x2, a2\}$	0.00000

For our running example, we are able to rule out one diagnosis, which is the largest one, and rank the other two, making use of passing and failing observations. The presented approach – of course – is based on some assumptions, i.e., (i) knowing the inputs and outputs of components and systems, (ii) being able to distinguish error prone from correct outputs, and (iii) the pragmatic choice of stating the relationship between the similarity coefficient and the fault probability. Some of these assumptions may not be achievable in certain applications.

7 Conclusions

In this paper, we discussed how multiple observation sets (like multiple test cases in the context of software engineering) can be integrated within the consistency-based diagnosis framework. The presented solutions are strongly based on previous research. Discussed extensions introduced in this paper are: (i) formalizing diagnosis when having multiple observation sets, (ii) coding this extended diagnosis problem using answer set programming, (iii) discussing limitations, and (iv) at least outlining the use of passing observations within the framework. The latter relies on earlier work on fault localization in spreadsheets. We provide a necessary extension, i.e., the use of multiple observation sets, and a link to the computation of diagnosis probabilities.

Future work has to include evaluating the approach within an experimental setup both considering systems comprising a larger set of components, and various observation sets. The experimental analysis shall also evaluate the use of similarity coefficients and their impact on the overall results. Furthermore, there are other ideas of integrating passing observations into diagnosis. For example, in [26] the author described the use of multiple observations to rule out diagnoses based on the assumption that components shall implement a function. Using this diagnosis and multiple observation sets, we may come up with an expect input output behavior of a certain component that is not a function. As a consequence, we may rule out this diagnosis. However, this can only be done assuming that in a faulty case, we still have a deterministic behavior of the involved components.

Acknowledgments

The authors want to thank the anonymous reviewers for their very valuable comments on the initial version of this paper. The research was supported by ECSEL JU under the project H2020 826060 AI4DI - Artificial Intelligence for Digitising Industry. AI4DI is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2019 and April 2022. More information can be retrieved from `https://iktderzukunft.at/en/`.

References

[1] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis based on structure and function. In *Proceedings AAAI*, pages 137–142, Pittsburgh, August 1982.

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

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

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

[5] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Hypothesis classification, abductive diagnosis and therapy. In *Proceedings of the International Workshop on Expert Systems in Engineering*, Vienna, September 1990. Springer Verlag, Lecture Notes in Artificial Intelligence, Vo. 462.

[6] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnosis and systems. *Artificial Intelligence*, 56, 1992.

[7] Luca Console and Pietro Torasso. Integrating models of correct behavior into abductive diagnosis. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 160–166, Stockholm, August 1990. Pitman Publishing.

[8] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Formalizing the repair process. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 709–713, Vienna, August 1992. John Wiley & Sons, Chichester. Also appeared in the Proceedings of the Second International Workshop on Principles of Diagnosis, Milano, 1991.

[9] Brian C. Williams and P. Pandurang Nayak. A Model-based Approach to Reactive Self-Configuring Systems. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis*, pages 267–274, 1996.

[10] Franz Wotawa. Reasoning from first principles for self-adaptive and autonomous systems. In E. Lughofer and M. Sayed-Mouchaweh, editors, *Predictive Maintenance in Dynamic Systems – Advanced Methods, Decision Support Tools and Real-World Applications*. Springer, 2019.

[11] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Physical impossibility instead of fault models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 331–336, Boston, August 1990. Also appears in Readings in Model-Based Diagnosis (Morgan Kaufmann, 1992).

[12] Iulia Nica, Ingo Pill, Thomas Quaritsch, and Franz Wotawa. The route to success - A performance comparison of diagnosis algorithms. In *IJCAI*, pages 1039–1045. IJCAI/AAAI, 2013.

[13] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *ASE*, pages 273–282. ACM, 2005.

[14] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J. C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[15] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, Aug 2009.

[16] Ingo Pill and Franz Wotawa. Computing multi-scenario diagnoses. In *Proceedings of the International Workshop on Principles of Diagnosis (DX)*, 2020.

[17] Paolo Ferraris and Vladimir Lifschitz. Mathematical foundations of answer set programming. In *We Will Show Them! (1)*, pages 615–664. College Publications, 2005.

[18] Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, volume 12144 of *LNAI*. Springer Nature Switzerland AG, 2020.

[19] David Kaufmann, Iulia Nica, and Franz Wotawa. Intelligent agents diagnostics – enhancing cyber-physical systems with self-diagnostic capabilities. *Advanced Intelligent Systems*, n/a(n/a):2000218, 2021.

[20] Alexey Ignatiev, Antonio Morgado, Georg Weissenbacher, and Joao Marques-Silva. Model-based diagnosis with multiple observations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, Macao, 2019.

[21] M. Kalech, R. Stern, and E. Lazebnik. Minimal cardinality diagnosis in problems with multiple observations. *Diagnostics*, 11(5), 2021.

[22] Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in gringo series 3. In *LPNMR*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer, 2011.

[23] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[24] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer-Verlag, 5th edition, 2003.

[25] Birgit Hofer, Alexandre Perez, Rui Abreu, and Franz Wotawa. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, pages 1–28, 2014.

[26] Franz Wotawa. Debugging hardware designs using a value-based model. *Appl. Intell.*, 16(1):71–92, 2002.