Abductive Diagnosis based on Modelica Models

Bernhard Peischl[1] and **Ingo Pill**[1] and **Franz Wotawa**[1*]
[1]TU Graz, Institute for Software Technology, Graz, Austria
e-mail: {bpeischl,ipill,wotawa}@ist.tugraz.at

Abstract

When employing model-based diagnosis, coming up with the required model is a knowledge and resource-intensive task. In this article, we show how to automatically derive an abductive diagnosis model from system models written in the popular Modelica modeling language. Using Modelica's simulation features we inject potentially faulty component behavior into the system model and automatically derive cause and effect rules. We afterwards use these rules for performing abductive diagnosis of the malfunctioning system. Our first case studies demonstrate the applicability and viability of the approach.

1 Introduction

A key element of any model-based diagnosis approach is the system model allowing us to reason about the correct functioning of individual system components [Davis, 1984; Reiter, 1987; de Kleer and Williams, 1987]. Unfortunately, the modeling task has turned out to be a stumbling block for a wide-spread use of model-based diagnosis in practice. That is, the additional resources needed for coming up with a diagnosis model, and the lack of options for integrating the modeling process for development purposes and the process of modeling for diagnosis purposes, have been countermanding the advantages of model-based approaches in many cases. In particular, there is a gap between models we create during development (e.g. for some simulation), and the models we require for diagnosing the deployed system. This gap can be explained with the different objectives behind the models. That is, e.g., a model used for simulation usually has to be detailed and as close to the system's real-world behavior as we need it to be for a certain simulation. In contrast, diagnosis models usually rely on qualitative system descriptions (for restricting the search space) such that several system behaviors which we do not need to distinguish for diagnosis purposes, are considered as a single qualitative one.

There are many languages available for modeling physical systems, including Modelica [Fritzson, 2014] which is an object-oriented, open, and multi-domain language. In Modelica, we define models via sets of equations that can range from simple algebraic equations to complex differential ones. These characteristics distinguish Modelica from other modeling languages, making it very flexible and easy to employ. In addition, there is a huge variety of available libraries targeting, e.g., digital circuits, electronics, mechanics, or fluids, making Modelica an ideal modeling language for a cyber-physical system (CPS). Modelica is optimized for simulation, so that any Modelica model has to ensure that its corresponding equations allow for computing exactly one solution, i.e., an assignment of variable values that solves all the equations at any point in time. Otherwise, an error message is raised. Consequently, we cannot use Modelica for diagnosis purposes directly, due to the lacking capabilities for dealing with unknown values or sets of values to be assigned to a variable.

Our motivation behind this paper is thus to provide a solution for automatically compiling Modelica models into ones that we can use for diagnosis. In the literature, there have been several other approaches dealing with this idea, e.g., [Sterling *et al.*, 2014; Minhas *et al.*, 2014; Matei *et al.*, 2015], as we discuss in detail in Section 2. In contrast to earlier work, we are interested in providing models to be used for abductive diagnosis, so that we extract cause-effect rules from Modelica models. Such rules are intuitive to designers familiar with failure mode and effect analysis (FMEA) [Hawkins and Woollons, 1998; Catelani *et al.*, 2010], making the approach even more attractive for practical purposes. The underlying idea of ours is to compare simulation results obtained for the *correct* model with results for *faulty* variants that we create via fault injection [Voas and McGraw, 1999]. In case of deviations, we introduce a rule stating that the introduced fault leads to the observed deviations. As we show in this paper, we then enable a designer to automatically come up with a set of cause-effect rules that she can use for abductive diagnosis.

This work extends previous work in the domain of abductive diagnosis [Wotawa, 2014; Christopher S. Gray and Wotawa, 2015] where we used FMEA like tables for extracting the cause-effect rules. In those papers, the authors introduced an algorithm that converts available tabular data on the available components, potential faults occurring for the individual components and the resulting effects, as well as supporting conditions into horn clauses for abductive diagnosis. The advantage of that approach is that it draws on information that is available in practice. Hence, an easy integration into existing processes for diagnosis and monitoring can be assured. On the downside, the tabular information has to be provided manually. Thus, in order to improve on this situation, in this paper we suggest to use popular modeling languages like Modelica for generating the desired ab-

[*]Authors are listed in alphabetical order.

ductive diagnosis models.

Before discussing our approach in detail in Section 3.2, let us briefly discuss the underlying ideas using the voltage-divider circuit shown in Figure 1. This voltage divider comprises two resistors and a battery, where if it works as intended, the battery's nominal voltage `BAT` of 12V is divided between resistors `R1` and `R2` to an 8V voltage drop for `R1` and a 4V voltage drop for `R2`. This, we can compute via Ohm's law $u = r * i$ (u being the voltage for some resistance r, and i being the current through r), and the individual resistance values of 100Ω and 50Ω for `R1` and `R2`. In Figure 1, we depict also some faulty behavior. In particular, we show how the circuit would behave if we assume that from time point 0.5 seconds onwards, either the battery is empty (bottom left picture) or resistor `R2` has a short (bottom right picture). In case of an empty battery, the voltage drops for both resistors are going down to 0V. For a short in `R2`, we have $v2$ =0V and $v1$ =12V, which means that the whole voltage of the battery is dropped on `R1`.

Such knowledge can be used to form an abductive diagnosis model. There we would state, for instance, that an empty battery causes both $v1$ and $v2$ to be 0V.

$$emptyBat \rightarrow val(v1, 0)$$
$$emptyBat \rightarrow val(v2, 0)$$

We might also want to express that in this case the voltage on both resistors is smaller than expected:

$$emptyBat \rightarrow smaller(v1)$$
$$emptyBat \rightarrow smaller(v2)$$

A similar model for a short resistor `R2` could look like:

$$short(\texttt{R2}) \rightarrow val(v1, 12)$$
$$short(\texttt{R2}) \rightarrow val(v2, 0)$$
$$short(\texttt{R2}) \rightarrow higher(v1)$$
$$short(\texttt{R2}) \rightarrow smaller(v2)$$

Any other fault mode, e.g., broken resistors et cetera, can be handled in a similar way. Now, if we have a simulation model available, so that we can simulate a fault's effects, we can automate this idea. Ideally, we would have a single simulation model, where we only need to switch on and off the individual faults to be considered in a simulation (switching between equations when doing so). When simulating the model with all faults turned off, and comparing the outcome with the output obtained for variants where we enable one individual fault after another, we are able to retrieve the desired information about the individual effects. As shown above, we can accommodate both, statements about the faulty behavior using concrete or maybe abstracted values, *and* also deviations between the expected correct and the experienced faulty value. The latter deviation model, i.e., a model where we state that a variable value would be smaller or higher than expected, has been used for diagnosis before, see, e.g., [Struss, 2004].

Before proposing our approach and its rule extraction algorithm in detail in Section 3.2, we discuss related literature in Section 2 where we focus on the use of Modelica for diagnosis. Afterwards, we introduce the preliminaries and other basic definitions in Section 3, in order for the paper to be self-contained. In Section 4 we discuss the automated rule extraction process in detail. Finally, we present some case studies in Section 5 and conclude in Section 6.

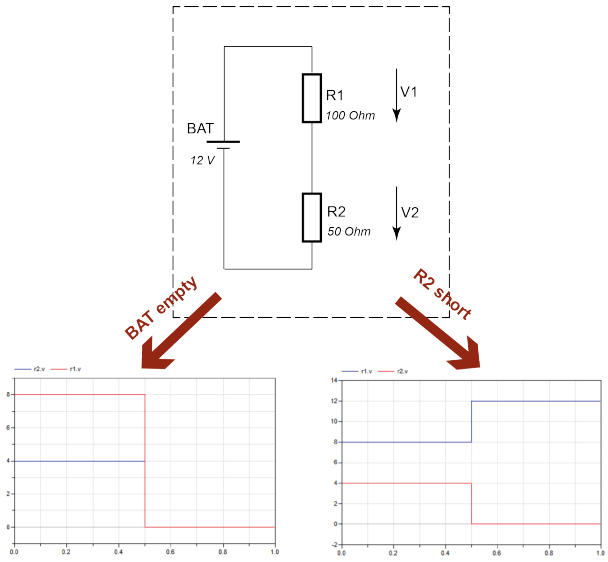

Figure 1: A voltage divider circuit

2 Related research

There have been three major approaches to using Modelica models for diagnosis. The first one implements the basic idea of applying those changes necessary for computing diagnoses to the language Modelica itself. This includes adaptions for handling unknown behavior, allowing us to come up with simulations where no single value can be determined anymore, and the introduction of corresponding fault modes and their behavior. In [Lunde, 2000], Lunde presented such an approach leading to the language Rodelica as used in the model-based diagnosis system RODON [Bunus *et al.*, 2009]. In contrast to [Lunde, 2000] we neither change the modeling language, nor do we rely on specific simulation engines. Rather, we make use of Modelica and its available simulation infrastructure, but of course assume that the components' specific fault modes are known and can be activated and deactivated during simulation.

The second approach augments Modelica models with fault modes and their behavior. In [de Kleer *et al.*, 2013; Minhas *et al.*, 2014] the authors correspondingly suggested to automatically augment Modelica models, and to use them for diagnosis as follows. When simulating the model with faults turned on or off, the outcome is compared with the expected fault free behavior. In case of differences that are considered to be large enough, the activated fault mode can be given back as result. The interesting idea behind [Minhas *et al.*, 2014] is that the authors suggest a Bayesian approach for checking similarity. This paper is very close to ours, with one important difference. The approach we introduce in this paper uses an augmented Modelica model for creating a knowledge-base for *abductive* reasoning, which can be used later on, i.e., after deployment, for diagnosis purposes.

The third approach to using Modelica for diagnosis (see [Sterling *et al.*, 2014]) uses corresponding models for computing a system's expected behavior to be compared with the actually observed and measured one. In case of a deviation, a model-based diagnosis engine is then used for computing explanations, i.e., diagnoses. The required model is extracted from the Modelica model such that basic components are replaced with qualitative models in order to come up with a component-connection model. Correspond-

ingly, we can use this concept for all cases where there are qualitative models available for all the library components contained in a model. An advantage of this approach is that we can use Modelica for obtaining a system's structure and for checking deviations between the expected and observed system output. A significant drawback is, however, that somebody has to develop the qualitative models for library components used in the system model.

There is further work dealing with diagnosis in the context of Modelica. For example, Lee et al. [Lee *et al.*, 2015] presented a very different and interesting approach. Also relying on component fault models, they use machine learning for identifying the root cause of an issue, rather than using a logic-based reasoning. In particular, the idea is to use simulation results from models where a component is assumed to be faulty for training a belief network. Once a certain behavior is observed, the network can then be used to isolate the corresponding faulty component. A related approach was presented in [Matei *et al.*, 2015] where the authors used partial models of a railway switch to learn diagnosis classifiers with a random-forest algorithm.

In contrast to Lee and colleagues, we rely on model-based diagnosis and in particular on abductive diagnosis. We furthermore do not rely on machine learning and we are also able to derive all diagnoses for a certain behavior, which is usually not possible when relying on belief networks.

3 Preliminaries

In this section, we define our underlying system model comprised of a system's Modelica model and augmented data like its list of components and their behavior modes. In the second half, we briefly introduce abductive diagnosis.

3.1 System modeling

Our starting point is a system's Modelica model. In the following definitions we state the data about such a model that we need for automatically extracting cause-effect rules based on fault injection [Voas and McGraw, 1999]. To this end, we assume that the Modelica model has the means for enabling or disabling individual fault modes as defined for the individual components. Every mode of a particular component can be a hypothesis for a certain observed faulty behavior, i.e., the observed symptoms. In addition, we differentiate between input and output variables, where the input variables are those for specifying a certain desired input behavior, and the outputs are those variables that might be observed and work as symptoms in the context of our abductive diagnostic reasoning.

Definition 1 (System model)**.** *A* system model $\mathbf{M}$ *is a tuple* $(COMP, MODES, \mu, I, O, \mathbf{P})$ *comprising a set of components COMP, a set of modes MODES that has at least the correct mode ok as element, a function* $\mu : COMP \mapsto MODES$ *mapping components to their featured modes, a set* I *of variables considered as inputs, a set* O *of variables considered as outputs, and a Modelica model* $\mathbf{P}$ *that allows for setting a mode* $m \in \mu(c)$ *for each component* $c \in COMP$*.*

Example 1. *In Fig. 2, we show the Modelica source code for our voltage-divider circuit example from Fig. 1. In this code, there is a general definition of a component* `MyComponent` *comprising some basic electrical behavior (e.g. knowledge concerning the pins).* `MyBattery` *and* `MyResistor` *are sub-classes of* `MyComponent`*, inheriting its equation, but extending it to capture the more specific behavior of the component type. The union of all fault modes for the individual components contains* `ok` *for the ordinary behavior,* `broken` *and* `short` *for a resistor's fault modes, and* `empty` *for a faulty battery. Hence, a* system model $(COMP, MODES, \mu, I, O, \mathbf{P})$ *would comprise the following elements:*

$$
\begin{aligned}
&COMP = \{\texttt{bat}, \texttt{r1}, \texttt{r2}\}\\
&MODES = \{ok, broken, short, empty\}\\
&\mu(\texttt{bat}) = \{ok, empty\}\\
&\mu(\texttt{r1}) = \mu(\texttt{r2}) = \{ok, broken, short\}\\
&I = \emptyset\\
&O = \{\texttt{r1.v}, \texttt{r2.v}\}
\end{aligned}
$$

■

In addition to a *system model*, we need to introduce the concept of simulation, i.e., the computation of values for a system's variables over time. To this end, we first introduce the concept of *mode assignments*, i.e., assignments of modes to components at particular points in time.

Definition 2 (Mode assignment)**.** *Let TIME be a finite set of time points. A mode assignment* Δ *is a set of functions* $\delta_i : COMP \times TIME \mapsto MODES$ *that assign for each component* $c \in COMP$ *and time point* $t \in TIME$ *a mode* $m \in \mu(c)$*, i.e.,* $\Delta = \{\delta_1, \ldots, \delta_{|TIME|}\}$ *where* $\forall i \in \{1, \ldots, |TIME|\} : \forall t \in TIME : \forall c \in COMP : \delta_i(c, t) \in \mu(c)$*.*

During simulation, a *mode assignment* allows for changing a component's behavior and in turn changing a system's behavior. In the following definition of a simulation function, we make use of mode assignments.

Definition 3 (Simulation)**.** *Let us assume that we have a* system model $\mathbf{M} = (COMP, MODES, \mu, I, O, \mathbf{P})$*, a test bench* $\mathbf{T}$ *specifying the desired system inputs over time, a mode assignment* Δ*, and an end time* t_e*. A* simulation *function* sim *is a function that computes via* $\mathbf{P}$ *the values of all variables over time between 0 and* t_e*, considering (a) test bench* $\mathbf{T}$ *for inputs* I *and (b) the mode assignment* Δ*.*

We can easily implement such a simulation function sim using a Modelica simulator. To this end, we construct a new test bench $\mathbf{T}'$ from $\mathbf{T}$ and Δ. A typical test bench $\mathbf{T}$ for a Modelica circuit `SUT` such that we change inputs over time, would follow, e.g., the following structure:

```
model Testbench
   SUT sys;
equation
  if (time < t1) then
    ....  // First inputs
  elsif (time >= t1 and time < t2) then
    .... // Next inputs
  elsif
    ....
  else
    ....
  end if;
end Testbench;
```

When taking Δ into account, we can easily extend $\mathbf{T}$ to derive $\mathbf{T}'$ that also captures the assignments of modes to the individual components over time.

Example 2. *Let us continue Example 1, considering the mode assignment* $\delta(\texttt{bat}, ok, 0)$*,* $\delta(\texttt{r1}, ok, 0)$*,* $\delta(\texttt{r2}, ok, 0)$*,* $\delta(\texttt{bat}, ok, 0.5)$*,* $\delta(\texttt{r1}, ok, 0.5)$*,* $\delta(\texttt{r2}, short, 0.5)$*. Due to the fact that the voltage divider has no input values, we can easily obtain test bench* $\mathbf{T}'$ *only considering* Δ*.*

```
connector MyPin
  Real v;
  flow Real i;
end MyPin;

type FaultType = enumeration(
     ok,broken,short,empty);

partial model MyComponent
  MyPin p,m;
  Real v;
  Real i;
  FaultType state(start = FaultType.ok);
equation
  v = p.v - m.v;
  i = p.i;
  0.0 = p.i + m.i;
end MyComponent;

model MyGround
  MyPin p;
equation
  p.v =0.0;
end MyGround;

model MyBattery
  extends MyComponent;
  parameter Real vn;
equation
  if state == FaultType.ok then
    v = vn;
  else
    v = 0.0;
  end if;
end MyBattery;

model MyResistor
  extends MyComponent;
  parameter Real r;
equation
  if state == FaultType.ok then
    v = r * i;
  elseif state == FaultType.short then
    v = 0;
  else
    i = 0;
  end if;
end MyResistor;

model MySimpleCircuit
  MyResistor r1(r=100);
  MyResistor r2(r=50);
  MyBattery bat(vn=12);
  MyGround gnd;
equation
  connect(bat.p,r1.p);
  connect(r1.m,r2.p);
  connect(bat.m,r2.m);
  connect(bat.m,gnd.p);
end MySimpleCircuit;
```

Figure 2: The Modelica program implementing the voltage divider circuit

```
model TestbenchPrime
   MySimpleCircuit sut;
equation
  if (time < 0.5) then
    sut.r1.state = FaultType.ok;
    sut.r2.state = FaultType.ok;
    sut.bat.state = FaultType.ok;
  else
    sut.r1.state = FaultType.ok;
    sut.r2.state = FaultType.short;
    sut.bat.state = FaultType.ok;
  end if;end Testbench;
```

■

Using a Modelica simulator and the extended test bench $\mathbf{T}'$, the simulation function sim can be defined as a call to this simulator using $\mathbf{P} \cup \mathbf{T}'$ and end time t_e as parameters. All values for outputs $o \in O$ in $\mathbf{M}$ will be computed during the simulation, and we assume that they are returned as a set of tuples (o, v) for t where $o \in O$ and v gives o's value.

3.2 Abductive diagnosis

In this subsection, we briefly recapitulate the basic definitions of *abductive diagnosis*. To this end, let us first introduce the concept of a *knowledge base*. A knowledge base comprises a set of horn clause rules *HC* over propositional variables *PROPS*. Please let us remind you that a horn clause is a clause (disjunction of literals) such that at most one literal is positive. When diagnosing engineered systems, such propositional variables state, for example, some component's particular mode or a certain value. Considering our example from the introduction, we might use, e.g., proposition $short(\texttt{R2})$ for stating that the resistor R2 from our running example has a short. A proposition $nok(\texttt{R2}.v)$ might be used to indicate that the voltage at resistor R2 deviates from the expected voltage. For stating a behavior, e.g., saying that a short of R2 leads to an unexpected voltage drop at the same resistor, we use the following horn clause $short(\texttt{R2}) \rightarrow nok(\texttt{R2}.v)$.

Now, using the definitions of [Friedrich *et al.*, 1990], let us introduce knowledge bases formally .

Definition 4 (Knowledge base (KB))**.** *A knowledge base (KB) is a tuple (A, Hyp, Th) where $A \subseteq$ PROPS denotes a set of propositional variables, $Hyp \subseteq A$ a set of hypotheses, and $Th \subseteq$ HC a set of horn clause sentences over A.*

In the context of this paper, hypotheses correspond directly to causes. Hence, from here on we will use the terms hypotheses and causes in an interchangeable way.

Example 3. *A partial KB for our running example as discussed before looks like:*

$$\begin{pmatrix} \{short(\texttt{R2}), nok(\texttt{R2}.v)\}, \\ \{short(\texttt{R2})\}, \\ \{short(\texttt{R2}) \rightarrow nok(\texttt{R2}.v)\} \end{pmatrix}$$

■

In the next step we define a *propositional horn clause abduction problem*.

Definition 5 (PHCAP)**.** *Given a knowledge base (A, Hyp, Th) and a set of observations $Obs \subseteq A$, the tuple (A, Hyp, Th, Obs) forms a propositional horn clause abduction problem (PHCAP).*

A solution of a PHCAP is a set of hypotheses that allows deriving the given observations or symptoms. The following definition from [Friedrich *et al.*, 1990] states this formally.

Definition 6 (Diagnosis; Solution of a PHCAP)**.** *Given a PHCAP* (A, Hyp, Th, Obs)*, a set* $\Delta \subseteq Hyp$ *is a solution if and only if* $\Delta \cup Th \models Obs$ *and* $\Delta \cup Th \not\models \bot$*. A solution* Δ *is* parsimonious *or* minimal *if and only if no set* $\Delta' \subset \Delta$ *is a solution.*

A solution Δ of some PHCAP is an *explanation* for the given observations. Thus we might also refer to Δ as *abductive diagnosis* (or diagnosis for short). In Definition 6, diagnoses do not need to be minimal or parsimonious. In most practical cases, however, only minimal diagnoses or minimal explanations for given effects are of interest. Hence, from here on, we assume that all diagnoses are minimal ones, if not specified explicitly otherwise.

Example 4. *Let us continue Example 3 and add the observation nok*$(\mathtt{R2}.v)$ *to the KB to form a PHCAP. The only solution for this problem is* $\{$*short*$(\mathtt{R2})\}$*. If we assume an observation* $\neg$*nok*$(\mathtt{R2}.v)$*, then there is no solution given the partial KB of Example 3.* ■

Finding minimal diagnoses for a given PHCAP is an NP-complete problem (see [Friedrich *et al.*, 1990]). However, computing all parsimonious solutions can be done easily and efficiently in cases where the number of hypotheses is not too big. An algorithm for computing abductive solutions might use De Kleer's Assumption-based Truth Maintenance System (ATMS) [de Kleer, 1986], where we refer the interested reader to [de Kleer, 1988] for an ATMS algorithm. For using an ATMS for abductive diagnosis, we only need to encode observations as a single rule, i.e., for observations $Obs = \{o_1, \ldots, o_k\}$, we generate a new proposition σ and add $o_1 \wedge \ldots \wedge o_k \rightarrow \sigma$ to the theory *Th* that is passed to the ATMS. The label of the corresponding node of σ is an abductive diagnosis for Obs. Due to the rules for the node labels, which only comprise hypotheses, it is ensured that the solution is minimal, sound, complete, and consistent. For more technical details, including computing distinguishing diagnoses and characterizing knowledge bases according to their capability of distinguishing diagnoses, we refer the interested reader to [Wotawa, 2014]. For a discussion on whether abductive reasoning can be used in practice, we recommend reading [Koitz and Wotawa, 2015].

4 Automated rule extraction

As briefly depicted in the introduction, we extract the desired rules from a system model $\mathbf{M}$ via comparing the outcome of two simulation runs, i.e., one assuming that no fault is enabled for any component, and one with exactly one fault mode enabled for one individual component. The difference between the two behaviors is mapped to a proposition, which can be used as observation in a PHCAP. The inputs are also mapped to propositions, as are those modes of components $c_i \in$ *COMP* that are not equal to the correct mode *ok*. The latter are also considered as hypotheses for diagnosis purposes.

Algorithm 1 formalizes the necessary steps for our rule extraction concept. In lines 1–2, we initialize the sets used in the computation, i.e., the set of propositions A, the set of hypotheses *Hyp*, and the horn clauses *Th*. We assume here that we have input values over time in I, as is necessary for stimulating the system under consideration. We further assume that this information can be mapped to a set of propositional variables and that it is also represented in the given test bench $\mathbf{T}$. A simple mapping would state that each variable occurring in I over time is ok, i.e., $ok(v, t)$ such that v is a variable and t a point in time where a value is set. Alternatively, we might state a precise certain value, which has to be provided in order to be able to use the extracted rules for diagnosis purposes, i.e., *value*(v, t, x) such that v is the name of the variable, t the time point, and x the value to be set for v at time t.

Algorithm 1 Rule extraction from Modelica models

Input: System model $(COMP, MODES, \mu, I, O, \mathbf{P})$, a test bench $\mathbf{T}$, a time t_f where a fault should be injected, and an end time t_e
Output: A KB (A, Hyp, Th)

```
1: Let I_p be the propositional representation of I.
2: Let A be I_p and let Hyp, and Th be empty sets.
3: for c in COMP do
4:     for m in μ(c) \ {ok} do
5:         Let Δ1(c, 0) = ok
6:         Let Δ2(c, 0) = ok and Δ2(c, t_f) = m
7:         for c' in COMP \ {c} do
8:             Δ1(c', 0) = ok
9:             Δ2(c', 0) = ok and Δ2(c'; t_f) = ok
10:        end for
11:        Let sim(P, T, Δ1, t_e) be B_corr.
12:        Let sim(P, T, Δ2, t_e) be B_faulty.
13:        Let D be the result of diff(B_corr, B_faulty) only
       considering variables in O.
14:        Add all elements of D to A.
15:        Add the proposition m(c) to A and Hyp.
16:        for d in D do
17:            Add the rule m(c) ∧ (⋀_{p∈I_p} p) → d to Th
18:        end for
19:    end for
20: end for
21: return (A, Hyp, Th)
```

Lines 3–20 implement the core of our rule extraction process. For all components and their modes we iterate (lines 3–4) over lines 5–17. In the first part of the inner loop's body (lines 5–10) the mode assignments for the correct simulation run (Δ_1) and for the faulty one (Δ_2) are generated. With the exception of the currently "active" component c (chosen in line 3), all other components are assigned mode *ok*. For component c, mode m (selected in line 4) is assigned for Δ_2 and *ok* for Δ_1 at a given point in time t_f. Note that here we assume a simpler mode assignment with only one permanent change at t_f. Future extensions will consider also different mode assignments over time. However, for our purpose, i.e., obtaining the effects of faults at specific points in time, this restriction seems to be appropriate. Note that t_f should be selected in a way such that the initialization of a system is finished and where we can observe the ordinary expected behavior.

In lines 11–13, we start the simulation runs and compare the observed results with function diff. In the simulation function sim, we assume that the test bench $\mathbf{T}$ is adapted such as to include mode assignment data (for switching fault modes of components on and off) like discussed and illustrated in the last section ($\mathbf{T} \rightarrow \mathbf{T}'$).

The function diff deserves special attention in that there are two potential concepts for implementing it. Either, in case that there is at least one output where the behaviors differ diff returns the propositional representation of values for all output variables from O, or diff returns the deviation of

values between $\mathbf{B}_{corr}$ and $\mathbf{B}_{faulty}$, e.g., stating that the value caused by an active fault for some observed variable $o \in O$ is smaller than the value expected for the case that there is no fault in the system at all. Please note that also Struss used the latter for diagnosis purposes [Struss, 2004].

The former mapping option requires a propositional representation for translating the real simulation values to a qualitative domain. For example, we might consider only values like 0, v_{max}, or v as qualitative values for a variable, such that 0 represents the zero value, v_{max} the maximum value that can be reached, and v any value between 0 and v_{max}. Obviously, the diagnosis capabilities might vary, depending on the chosen qualitative representation and the encoded information. In [Sachenbacher and Struss, 2003; 2005], the authors discuss this issue, i.e., the task of finding an appropriate task dependent qualitative abstraction, and also show how to automate this abstraction. Now let us formalize these two approaches at implementing diff:

Qualitative representations: For a qualitative representation, we assume a quantitative domain D and its qualitative representation D_Q, together with a mapping function $\rho : D \mapsto D_Q$. Then, we define diff as:

$$\mathsf{diff}(B_1, B_2) = \begin{cases} \emptyset & \text{if } \nexists (x,v) \in B_1, (x,v') \in B_2 : v \neq v' \\ \{val(x, \rho(v)) | (x,v) \in B_2\} & \text{otherwise} \end{cases}$$

Deviation models: In case we prefer a deviation model, we are "only" interested in whether the value for a variable is, e.g., smaller, equal, or larger than its expected value. Hence, we define diff straightforwardly as follows:

$$\mathsf{diff}(B_1, B_2) = \{o(x) | (x,v) \in B_1, (x,v'') \in B_2 \wedge v' \; o^* \; v\}$$

In this definition, o and o^* represent the relational operator on the side of the qualitative and quantitative domain respectively. For example, the operator $o \in \{smaller, equal, larger\}$ in the qualitative domain corresponds to $o^* \in \{<, =, >\}$ in the quantitative (integer, real) domain. While we showed the obvious relational operators that we might want to consider, for some projects, we might also want to add, e.g, ones indicating that there is a huge difference in the values (above a certain threshold).

In the last part of the core functionality of our rule extraction algorithm (i.e., lines 14–18), we add the obtained propositions, hypotheses, and horn clause rules to the respective sets. Finally, we return the PHCAP in line 21.

It is easy to see that, by construction, our rule extraction algorithm works as expected. If we assume that sim and diff terminate, then termination is ensured since we only have a finite number of components and modes. In respect of time, the complexity of Algorithm 1 is bounded by $O(|\mathit{COMP}|^2 \cdot |\mathit{MODES}|)$ when assuming that sim and diff run in unit time. Note that in practice simulation is very much likely to be responsible for most of the experienced run-time. However, since we can automate all parts of the algorithm and can execute them before deploying the system (and in turn before deploying the diagnosis engine), the time complexity of the rule conversion process seems to be negligible since we do not encounter it when running the diagnosis engine itself.

5 Case studies

In order to show the viability of our proposed method, we carried out two case studies. So let us start with the *first case study* showing the results we obtained when using Algorithm 1 for our voltage-divider example. The corresponding source code for the Modelica model is shown in Figure 2, and the list of components *COMP*, the list of behavioral modes *MODES*, etc. were derived in Examples 1 and 2. When applying Algorithm 1 considering $\mathit{COMP} = \{\mathtt{bat}, \mathtt{r1}, \mathtt{r2}\}$, $\mathit{MODES} = \{ok, broken, short, empty\}$, and assuming t_f to be set to 0.5 seconds like for Example 2, we would obtain the following results when simulating the resulting faulty behavior.

Component	Mode	v_1	v_2	e_1	e_2
BAT	*empty*	0	0	smaller	smaller
R1	*short*	0	12	smaller	larger
R1	*broken*	12	0	larger	smaller
R2	*short*	12	0	larger	smaller
R2	*broken*	0	12	smaller	larger

In this table, we also state the deviations (or effects e_1, e_2) for variables v_1 and v_2 and the individual fault modes (let us remind you that the nominal values should be 8V for v_1 and 4V for v_2). What we see also from our example is that the values obtained when simulating the Modelica program, do not necessarily guarantee that we are able to distinguish between all diagnoses. For example, both a *broken* R1 and a *short* R2 would produce the same values for v_1 and v_2, so that we do not end up with a single explanation/diagnosis in general. Let us now use this table to generate the qualitative and the deviation model (concerning diff in Algorithm 1) for the voltage divider. For both models we have the same set of hypotheses:

$$\left\{ \begin{array}{c} empty(\mathtt{BAT}), short(\mathtt{R1}), broken(\mathtt{R1}), \\ short(\mathtt{R2}), broken(\mathtt{R2}) \end{array} \right\}.$$

Qualitative model For this kind of model we assume a qualitative domain of $\{0, (0,4), 4, (4,8), 8, (8,12), 12\}$ such that we consider all voltage values occurring in the correct model. The open interval (x, y) stands for any value larger than x and smaller than y. For this representation, the algorithm would return the following rules:

$$\begin{array}{c} empty(\mathtt{BAT}) \rightarrow val(v1, 0) \\ empty(\mathtt{BAT}) \rightarrow val(v2, 0) \\ short(\mathtt{R1}) \rightarrow val(v1, 0) \\ short(\mathtt{R1}) \rightarrow val(v2, 12) \\ broken(\mathtt{R1}) \rightarrow val(v1, 12) \\ broken(\mathtt{R1}) \rightarrow val(v2, 0) \\ short(\mathtt{R2}) \rightarrow val(v1, 12) \\ short(\mathtt{R2}) \rightarrow val(v2, 0) \\ broken(\mathtt{R2}) \rightarrow val(v1, 0) \\ broken(\mathtt{R2}) \rightarrow val(v2, 12) \end{array}$$

For this case, the set of propositions includes the hypotheses (like $empty(\mathtt{BAT})$ and the elements in $\{val(v1, 0), val(v1, 12), val(v2, 0), val(v2, 12)\}$. No other qualitative values are necessary for this example.

Deviation model The deviation model for our running example comprises the following rules:

$$
\begin{array}{c}
empty(\texttt{BAT}) \rightarrow smaller(v1) \\
empty(\texttt{BAT}) \rightarrow smaller(v2) \\
short(\texttt{R1}) \rightarrow smaller(v1) \\
short(\texttt{R1}) \rightarrow larger(v2) \\
broken(\texttt{R1}) \rightarrow larger(v1) \\
broken(\texttt{R1}) \rightarrow smaller(v2) \\
short(\texttt{R2}) \rightarrow larger(v1) \\
short(\texttt{R2}) \rightarrow smaller(v2) \\
broken(\texttt{R2}) \rightarrow smaller(v1) \\
broken(\texttt{R2}) \rightarrow larger(v2)
\end{array}
$$

In this case, the set of propositions is formed by the hypotheses together with propositions $\{smaller(v1), larger(v1), smaller(v2), larger(v2)\}$.

Both kinds of model represent the obtained information in a qualitative way. It is worth mentioning, that in addition, we might also want to encode in a knowledge base that some values cannot occur at the same time. For example, a voltage drop cannot be larger *and* smaller than some value at the same time. Such knowledge can be added easily via stating that $\textit{smaller}(v1) \wedge \textit{larger}(v1) \rightarrow \bot$ and $\textit{smaller}(v2) \wedge \textit{larger}(v2) \rightarrow \bot$. Of course, we can also automate the process of adding such mutual exclusiveness data.

The purpose of the *second case study* is to show that our approach can be applied also to to analog circuits comprising capacitors and switches such that the behavior over time is more complicated. In Figure 3, we show such a circuit, where we use a switch `SW` for turning a bulb `BULB` on or off. The purpose of capacitor `C1` is such that the bulb stays on (red line in Fig. 4) for a short while after switching it off (see blue line in Fig. 4), drawing from the energy stored while loading `C1` - see the green line for the current in the capacitor). Thus, we have one input, i.e., `SW`, and one output, i.e., `BULB` transmitting light or not. Appropriate sets of behavioral modes would be $\{ok, empty\}$ for the battery, and $\{ok, short, broken\}$ for the other components. Via corresponding simulations, we obtained the results shown in the following table, where we focus on the first difference between the expected value of *light* and the observed one at a particular point in time. In the table we see also the value of the input variable *on* stating the status of `SW`.

Component	Mode	*on*	*light*	*time*
BAT	*empty*	on	off	0.5
R1	*short*	off	on	1.13
R1	*broken*	on	off	0.5
SW	*short*	off	on	0.0
SW	*broken*	on	off	0.5
C1	*short*	off	off	1.0
C1	*broken*	off	off	1.0
R2	*short*	off	off	1.09
R2	*broken*	off	off	1.0
BULB	*short*	on	off	0.5
BULB	*broken*	on	off	0.5

Again, it is evident from the table that we cannot distinguish between all faults via the obtained deviations. We also see from Figure 4 that for a short in `R1`, the output might have the same shape, but would show a slightly different timing. Using the obtained table of behavioral deviations, computing the knowledge base for abductive diagnosis as described in this paper is straightforward. However, for more detailed results, we should probably introduce the means for reasoning about differences in the timing, e.g., stating that `BULB` goes off too late. Thus, we see that our approach for extracting a PHCAP can be used for many systems, and that the detail level of the obtained knowledge base depends on the chosen deviation description as expected and discussed.

6 Conclusion

When facing the decision of whether to employ some model-based diagnosis approach for a project, more often than not the resources and knowledge to come up with the needed model prohibit a decision towards an implementation. In our paper, we address some of the issues that make the required modeling step so demanding. That is, we show for a popular modeling language how to *automatically* derive a model containing cause and effect rules that we can then use for abductive diagnosis. For computing our rules, we make use of Modelica's simulation engine, where the only data a designer has to deliver are data that she would consider during FMEA anyway. This data concerning fault models for components and input vectors for triggering them can, which is important, be defined in Modelica. In particular, a designer can add multiple behaviors for a component to the model, and our approach will enable them individually in order to simulated the corresponding behavior as basis for our diagnosis model.

The derived abductive diagnosis model can then be used to solve propositional horn clause abduction problems which basically consist of the data we derived when creating our cause and effect rules plus the observed symptoms. An advantage of this type of model-based reasoning is that it is quite intuitive to what a maintenance expert would think about when considering FMEA data, but offers the advantage of a formal background and thus is amenable to automated reasoning.

Currently, we have been limiting the algorithm to the simulation of single faults. For future work, we intend to extend this to simulations of scenarios with multiple faults. Like we mentioned, we plan to extend the algorithm also to incorporate a more general intermediate fault activation as hinted at in our definitions. Last but not least, while we showed the viability of our approach for some examples, we need more and also industrial sized case studies as showcases. Hopefully corresponding results can contribute to increasing the deployment of model-based diagnosis in practice.

Acknowledgments

The work presented in this paper has been supported by the FFG project Applied Model Based Reasoning (AMOR) under grant 842407 and SFG project EXPERT. We would further like to express our gratitude to our industrial partner Uptime Engineering GmbH.

References

[Bunus *et al.*, 2009] Peter Bunus, Olle Isaksson, Beate Frey, and Burkhard Münker. Rodon - a model-based diagnosis approach for the DX diagnostic competition. In *Int. Workshop on Principles of Diagnosis (DX)*, 2009.

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

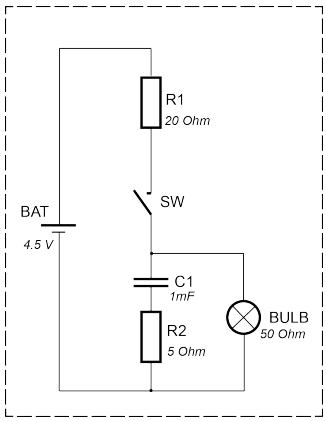

Figure 3: A switch circuit example

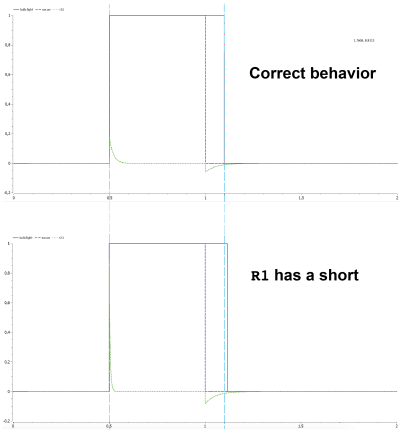

Figure 4: The correct behavior (picture at the top) versus the behavior in case short `R1`

[Christopher S. Gray and Wotawa, 2015] Siegfried Psutka Christopher S. Gray, Roxane Koitz and Franz Wotawa. An abductive diagnosis and modeling concept for wind power plants. In *9th IFAC symposium on fault detection, supervision and safety of technical processes*, 2015.

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

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

[de Kleer *et al.*, 2013] Johan de Kleer, Bill Janssen, Daniel G. Bobrow, Tolga Kurtoglu, Bhaskar Saha, Nicholas R. Moore, and Saravan Sutharshana. Fault augmented Modelica models. In *24th Int. Workshop on Principles of Diagnosis (DX)*, pages 71–78, 2013.

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

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

[Friedrich *et al.*, 1990] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Hypothesis classification, abductive diagnosis and therapy. In *International Workshop on Expert Systems in Engineering*, 1990.

[Fritzson, 2014] Peter Fritzson. *Object-Oriented Modeling and Simulation with Modelica 3.3 – A Cyber-Physical Approach*. Wiley-IEEE Press, 2 edition, 2014.

[Hawkins and Woollons, 1998] 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.

[Koitz and Wotawa, 2015] R. Koitz and F. Wotawa. On the feasibility of abductive diagnosis for practical applications. In *9th IFAC symposium on fault detection, supervision and safety of technical processes*, 2015.

[Lee *et al.*, 2015] Dongkyu Lee, Byoungdoo Lee, and Jin Woo Shin. Fault detection and diagnosis with modelica language using deep belief network. In *Proceedings of the 11th International Modelica Conference*, pages 615–623, Versailles, France, September 21-23 2015.

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

[Matei *et al.*, 2015] Ion Matei, Anurag Ganguli, Tomonori Honda, and Johan de Kleer. The case for a hybrid approach to diagnosis: A railway switch. In *26th Int. Workshop on Principles of Diagnosis*, pages 225–234, 2015.

[Minhas *et al.*, 2014] Raj Minhas, Johan de Kleer, Ion Matei, and Bhaskar Saha. Using fault augmented modelica models for diagnostics. In *10th International Modelica Conference*, March 10-12 2014.

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

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

[Sachenbacher and Struss, 2005] Martin Sachenbacher and Peter Struss. Task-dependent qualitative domain abstraction. *Artif. Intell.*, 162(1-2):121–143, 2005.

[Sterling *et al.*, 2014] Raymond Sterling, Peter Struss, Jesus Febres, Umbreen Sabir, and Marcus M. Keane. From modelica models to fault diagnosis in air handling units. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, March 10-12 2014.

[Struss, 2004] Peter Struss. Deviation models revisited. In *Working Papers of the 15th Int. Workshop on Principles of Diagnosis (DX-04)*, 2004.

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

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