# Reasoning from First Principles for Self-adaptive and Autonomous Systems


**Franz Wotawa**

## 1 Introduction

Systems that are able to automatically adapt their behavior in cases of faults have always been very much appealing for research and practice. Because of the increasing importance of applications like autonomous vehicles, the internet of things (IoT), or industry 4.0 dealing with increased autonomy, self-adaptation increases importance as well. For example, consider a truly autonomous vehicle transporting passengers from one location to another. If there is a system fault occurring during operation, there is no human driver working as fallback mechanism for assuring that the vehicle goes to a safe state, e.g., driving to an emergency lane of a highway and stopping there. In case of autonomous driving the system itself is responsible for any action after detecting a failure. This is one of the most significant differences to ordinary cars even if they have implemented automated functions like lane assist.

It is also worth noting that in many situations coming to a safe state is not that simple even for today's cars. For example, an emergency break as consequence of a fault in the control system of the car's engine might cause an accident if this is done on a high way with another car behind. Or another example is stopping a car in a tunnel without an emergency lane. Therefore, a car that is not moving is not necessarily in a safe state. As a consequence faults during operation should be handled in a smart way either via compensating or repairing faults during operation requiring self-adaptive systems. Of course it is worth mentioning that such a self-adaptive behavior does not allow to compromise safety. According to the IEEE

F. Wotawa (✉)
Technische Universität Graz, Institute for Software Technology, Graz, Austria
e-mail: wotawa@ist.tugraz.at



E. Lughofer, M. Sayed-Mouchaweh (eds.), *Predictive Maintenance in Dynamic Systems*, https://doi.org/10.1007/978-3-030-05645-2_15

Systems and Software Engineering Vocabulary [29] fail-safe refers to a system or component that automatically places itself in a safe operating mode in the event of a failure.

Self-adaptive systems that assure fail-safe behavior in the context of autonomous vehicles are also often referred to fail-operational system, i.e., system that still remain operational even in case of faults. In order to implement fail-operational behavior we have to use methods that assure also the behavior to be fail-safe. We need methods that can be proven to work as expected never compromising safety. We therefore discuss methods relying on model-based reasoning in this chapter, because such methods guarantee to deliver all results that fulfill all properties specified in models. What remains is to prove that the models capture the important parts of the system and also its properties like safety.

The idea of using models for various purposes like diagnosis is not new and dates back to the early 1980s of the last century (see, for example, [11]). Model-based reasoning is characterized of using models directly to implement certain tasks without requiring to reformulate available knowledge. In case of diagnosis, the model is used to derive the expected behavior that can be compared with observations. If we see a deviation between the expected behavior and the observed one, model-based reasoning utilizes the model to identify the root cause of the detected misbehavior. The basic principles behind model-based reasoning are still very suitable for today's challenges like autonomous systems and driving. If a model is appropriately capturing its corresponding system, then all conclusions drawn from the model are reasonable and also appropriate. Therefore, validation and verification can focus on the model once the reasoning algorithms are tested.

In this chapter we discuss the basic principles of model-based reasoning including algorithms. We do not only focus on one available technique that makes use of models formalizing the correct behavior of components, but also abductive diagnosis where we use fault models for obtaining root causes in a similar way than medical doctors do when reasoning from observed symptoms back to hypotheses. In addition, we outline an approach for online repair of systems interacting with their environment using sensors and actuators. We discuss how to integrate diagnosis with repair and also the different types of repair. For the latter, we make use of a running example from the autonomous robotics domain. It is worth noting that the purpose of the chapter is mainly to give an overview of model-based reasoning for self-adaptive systems. Therefore, we also discuss related research and previous work that has been published.

This chapter is organized as follows: We first introduce the application domain of autonomous mobile robots in Sect. 2. Afterwards, in Sect. 3 we introduce the basic concepts of model-based reasoning including model-based diagnosis and abductive diagnosis. In Sect. 4 we discuss issues of modeling for model-based reasoning, followed by Sect. 5 where we introduce the basic architectures behind a self-adaptive system. There we make use of three examples outlining

the different repair actions necessary to bring the system back into an operational state. Finally, we discuss related literature in Sect. 6 and conclude the chapter in Sect. 7.

## 2 Example

In this section, we introduce the example of a mobile robot having a differential drive for moving from one point in a plain to another. We will use this example in our chapter for introducing the basic concepts behind model-based reasoning and in an extended form for showing how self-adaptive behavior can be implemented using models of the system directly.

A differential drive comprises two wheels with varying speed. Depending on the speed of the wheels the robot either rotates, moves on a straight line, or on a curve. In the following, we discuss a kinematics model of a mobile robot with a differential drive. For more details we refer the interested reader to [17]. In Fig. 1 we show the underlying ingredients. We assume that the robot is at its current position $(x_R, y_R)$ heading in a direction specified by the angle $\theta$ from the $x$-axis. We further assume that the distance between the wheels is $d$. Depending on the speed of the right or left wheel $v_R$, $v_L$, respectively, the robot rotates about its instantaneous center of curvature (ICC) with a rotational speed $\omega$. The ICC lies on a straight line between the axis of the wheels and its distance from the center of the robot (lying on the same line) is $R$.

Obviously, there must be a relationship between $\omega$ and the speed of the wheels because both wheels are on the same line connected with ICC and thus have to have the same rotational speed. We are able to formalize this relationship as follows:

$$\begin{aligned} \omega(R + d/2) &= v_R \\ \omega(R - d/2) &= v_L \end{aligned} \tag{1}$$

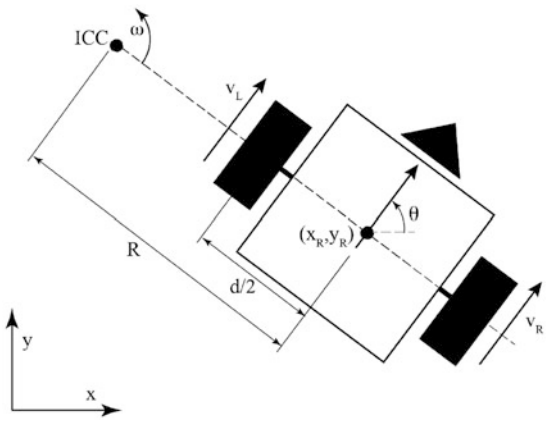


**Fig. 1** Mobile robot with differential drive

From Eq. (1) we are able to obtain $R$ and $\omega$ if knowing $v_R$ and $v_L$.

$$R = \frac{d}{2}\frac{v_R + v_L}{v_R - v_L}; \qquad \omega = \frac{v_R - v_L}{d} \tag{2}$$

From Eq. (2) we can distinguish 3 corner cases of movement for a differential drive robot, which are usually used as available actions when planning a route for the robot from its current position to its finally expected position:

1. If $v_R = v_L > 0$, then $\omega$ becomes zero, and $R$ infinite. Hence, the robot is moving on a straight line.
2. If $v_R = -v_L$, then $R$ becomes zero, and we obtain a rotation around the center of the robot. The direction of the rotation in this case is clockwise (assuming $v_L > 0$) and counter clockwise, otherwise.
3. If $v_L = 0$ and $v_R > 0$ ( $v_L > 0$ and $v_R = 0$), then $R = \frac{d}{2}$ ($R = -\frac{d}{2}$) and the robot rotates counter clockwise on its left wheel (clockwise on its right wheel).

Based on the above equations, we are also able to come up with a forward kinematics of the differential drive robot. In this case we assume that the robot is at a specific position $(x_R, y_R)$ and direction with angle $\theta$. The speed $v_R$ and $v_L$ are the control parameters to bring the robot to a new position. Using Eq. (2) we first obtain the ICC location:

$$\text{ICC} = (x - R\sin(\theta), y + R\cos(\theta)) \tag{3}$$

Assuming the robot is at its location at time $t$ we are now able to state its new position at time $t + \Delta t$ where $\text{ICC}_x$ and $\text{ICC}_y$ references are the ICC location of the $x$- and $y$-axis, respectively:

$$\begin{pmatrix} x'_R \\ y'_R \\ \theta' \end{pmatrix} = \begin{pmatrix} \cos(\omega\Delta t) & -\sin(\omega\Delta t) & 0 \\ \sin(\omega\Delta t) & \cos(\omega\Delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_R - \text{ICC}_x \\ y_R - \text{ICC}_y \\ \theta \end{pmatrix} + \begin{pmatrix} \text{ICC}_x \\ \text{ICC}_y \\ \omega\Delta t \end{pmatrix} \tag{4}$$

Using Eq. (4) we are able to predict the movements of a robot with a differential drive over time providing that we know the speed of the wheels $v_R$ and $v_L$. In control engineering someone would also be interested to compute values for $v_R$ and $v_L$ in order to reach a certain goal location. Searching for such values is also known as inverse kinematics problem. In case of a differential drive we are not able to compute such velocities. Instead what we can do is to separate this problem. We are able to move on a straight line and we are also able to rotate the robot on its current place. Hence, the problem of reaching an arbitrary location can be solved rotating the robot such that there is only a straight movement necessary to reach the goal, and afterwards move forward.

Without any doubt the given equations provide a model that explains the kinematics of a differential drive robot providing that the relevant information like the wheel

speeds, the robot's dimensions, and the current location of a robot together with its direction is known. However, in the following, we will not use those equations directly. Instead we will focus on an abstraction of the behavior for diagnosis and also for implementing self-healing behavior. The abstraction we are going to use can be easily obtained from the cases we distinguished for the robot's movement providing the wheel speeds. For example, we might consider a finite number of values for speed, i.e., either the speed is 0 or the positive or negative nominal value. In this case the domain would be $\{v_n^-, 0, v_n^+\}$. Using this domain we are able to formalize the ordinary behavior of a differential drive robot using first order logic (FOL) as follows:

$$val(v_L, v_n^+) \wedge val(v_R, v_n^+) \rightarrow motion(straightline)$$
$$val(v_L, v_n^+) \wedge val(v_R, v_n^-) \rightarrow motion(rotateclockwise)$$
$$val(v_L, v_n^-) \wedge val(v_R, v_n^+) \rightarrow motion(rotatecounterclockwise)$$
$$val(v_L, 0) \wedge val(v_R, 0) \rightarrow motion(stop)$$

In the rules we use the predicate $val/2$ stating that a particular speed given as first argument has the value given in the second argument. The predicate $motion/1$ is for establishing that a certain motion pattern is valid, i.e., either following the straight line or rotating clockwise or counter clockwise. The first rule formalizes the case where a robot moves on a straight line. The second is for indicating the case of clockwise rotation. The third one is for rotating counter clockwise, and the last one specifies the case where the robot stops moving. Note that this formalization does not comprise the case where the speed of one wheel is set to a value unequal 0, and the one of the other wheel is set to 0. Such a setting would also lead to a rotation and can be easily added to the abstract model if required.

The control problem of reaching a certain location can be represented using abstract values for the speed of the wheels. The following sequence assures that the robot first rotates and afterwards moves straightforward.

$$(v_L = v_n^+, v_R = v_n^-)_0, (v_L = v_n^+, v_R = v_n^+)_1$$

In this representation, we use $(\ldots)_i$ to indicate given values to be used at time $i$. Note that we do not consider a specific time. Instead each element indicates a state occurring at a particular point in time and lasting for a certain period. Hence, the presentation abstracts not only the values but also time. In the following section we show how such abstract models can be used for identifying the cause of a detected misbehavior. For example, we will consider the case of a differential drive robot that follows a wrong trajectory. In Fig. 2 we depict such a case, where a robot is expected to rotate clockwise followed by moving on a straight line but follows a curve. Obviously in this case either the speed of the left wheel is too low or the one of the right wheel too high.

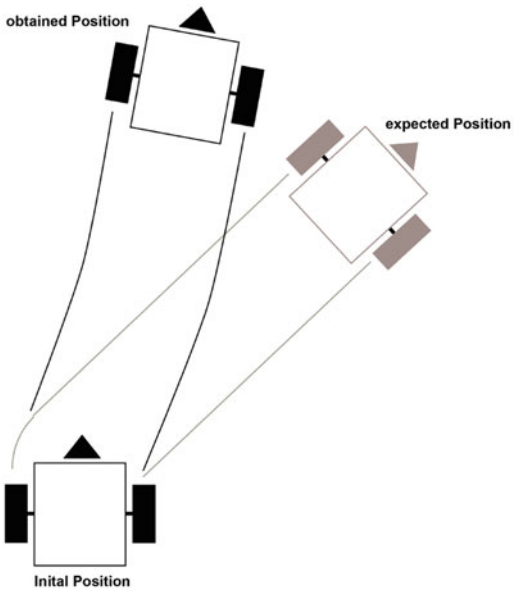


**Fig. 2** A small mobile robot driving the wrong trajectory

## 3 Model-Based Reasoning

The underlying idea behind model-based reasoning is to use a model of a system directly to reason about the system. In one instance, i.e., model-based diagnosis, the system's model is used for identifying root causes in case of an observed behavior that contradicts the expected one. A model in model-based diagnosis (MBD) comprises the system's structure including its components and interconnections, as well as the component models. The health state of components, i.e., a predicate indicating whether a component is working as expected or not, is used to indicate a root cause. In this terminology an incorrectly working component maybe an explanation for the detected unexpected deviation in the observed behavior. Davis [11] was one of the first outlying basic principles behind MBD that Reiter [46] and De Kleer and Williams [13] further formalized and extended.

It is worth noting that in classical MBD the component models only describe the correct behavior. This makes the theory general applicable even in cases where there is no knowledge about faults and their consequences available. De Kleer et al. [14] later presented an extension incorporating models of faulty behavior into the theory including some theoretical consequences. For example, in MBD without fault models every superset of a diagnosis itself is a diagnosis, which is not the case when using fault models. Note that there is a close relationship between MBD and other diagnosis theories like abductive diagnosis [21]. In abductive diagnosis, symptoms are explained based on hypotheses, which—more or less—represent known faulty behavior. Console and Torasso [7] showed how to integrate also correct behavior into abductive reasoning, and later Console et al. [8] showed that abductive diagnosis is MBD using models of faulty behavior.

In this section, we recall the basic foundations behind MBD and also abductive diagnosis. For this purpose, we make use of the differential drive robot as a running example. In contrast to the mathematical model of the kinematics outlined in Sect. 2, we will use an abstract representation, which we initially discussed in the same section. Because of the fact that diagnosis relies on systems comprising interconnected components, we first start with such a component-oriented model for a differential drive robot. In Fig. 3 we depict such a robot where each wheel has a wheel encoder attached and is connected to an electric motor that drives the wheel. The wheel encoder is for giving feedback to a controller that supplies the motors of the left and the right wheel with their expected voltage level.

In Fig. 4 we summarize the component-oriented representation of the differential drive robot. The control component $C$ is connected to motor $M_L$ and $M_R$. The motors are connected to their corresponding wheels $W_L$ and $W_R$, respectively. With attached wheel encoders $E_L$ and $E_R$ the rotational speed of the wheels is given back to the controller $C$. Hence, the motors work as actuators whereas the wheel encoders as sensors.

The behavior of each component can be specified again in an abstract way using the value domain $\{v_n^-, 0, v_n^+\}$ where distinguish a negative nominal value, zero, and

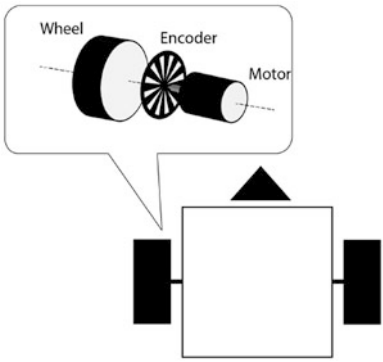


**Fig. 3** A small mobile robot comprising a differential drive with two motors and two wheel encoders for obtaining the wheels' rotation

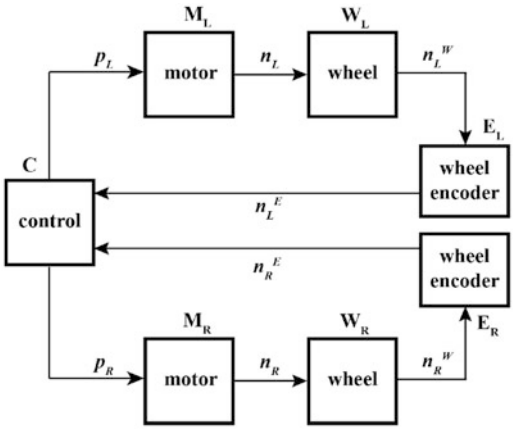


**Fig. 4** The component-oriented model of the differential drive robot

a positive nominal value, respectively. For example, if the input to the motor is a positive nominal value, then its output, i.e., its rotational speed, is also a positive nominal value. We can similarly define the behavior of a wheel and the wheel encoders. We will further formalize the components' behavior when introducing MBD and afterwards abductive diagnosis in the following subsections.

### *3.1 Model-Based Diagnosis*

In this subsection we outline the basic definitions of MBD from Reiter [46] in slightly adapted form. According to Reiter, MBD allows to reason directly from models and is therefore also called reasoning from first principles. A system model itself comprises components, their interconnections, and the components' behavior. All components of the system that might cause a misbehavior are assumed to be element of a set $COMP$. The structure and behavior has to be specified in a formal form in $SD$. Formally, a system (model) according to Reiter is defined as follows:

**Definition 1 (Diagnosis System)** A pair $(SD, COMP)$ is a diagnosis system providing that $SD$ is a system description comprising a model of the system, and $COMP$ a set of system components.

Using Definition 1 we are able to represent the differential drive robot as diagnosis system as follows: We start with the components. In the representation we only take care of components, which we want to classify as faulty or correct. Hence, in this example, we only consider the motors and the wheel encoders to be faulty, ignoring the health state of the control component and the wheels, so that:

$$COMP_R = \{M_L, M_R, E_L, E_R\}$$

Despite this design decision, we have to formulate a model of the wheels as well. Basically, we have to state that if there is a rotation applied at the axis, it is also provided to the wheel encoder. Using FOL, we are able to express this behavior as follows:

$$\forall X : wheel(X) \rightarrow (\forall Y : dom_A(Y) \rightarrow (val(in(X), Y) \leftrightarrow val(out(X), Y)))$$

In the above rule we make use of a predicate $value/2$ to assign a value to a port of a component, where we assume that a wheel has only one input and one output. We further restrict the values using the predicate $dom_A/1$ to three values as follows representing the abstract value domain introduced in Sect. 2.

$$dom_A(v_n^-) \wedge dom_A(0) \wedge dom_A(v_n^+)$$

Note that in this domain we only consider nominal speed and speed 0. If needed in an application scenario, we might either use more abstract values or to use even models based on the continuous domain. In the former case, it is important to consider all abstract values that allow us to distinguish the different behavior of

a system we want to diagnose. In the latter case, we might have to use a different underlying reasoning system for diagnosis.

In addition to the domain description, we add the information that we have two wheels to the model stating:

$$wheel(W_L) \wedge wheel(W_R).$$

The models for the motors and the encoders can be similarly formalized. In both cases there is only one input and one output. If the input is zero, then the output has also to be zero. If it is a nominal value, the value is propagated to the output. However, in contrast to the wheel, we now have the situation that a component might fail. In this case we do not know its behavior. In order to distinguish the health state of a component, we use a new predicate $Ab/1$ for each element of $COMP_R$ that if true, states that the component is faulty. For MBD we only specify the correct behavior requiring to formalize a rule in case $\neg Ab$ is true. For the motor and the encoder, we use the following rules for this purpose:

$$\begin{aligned}\forall X : motor(X) \rightarrow (\forall Y : dom_A(Y) \rightarrow (\neg Ab(X) \\ \rightarrow (val(in(X), Y) \leftrightarrow val(out(X), Y)))) \\ \forall X : enc(X) \rightarrow (\forall Y : dom_A(Y) \rightarrow (\neg Ab(X) \\ \rightarrow (val(in(X), Y) \leftrightarrow val(out(X), Y))))\end{aligned}$$

Again, we also represent the structure of the system formally:

$$motor(M_L) \wedge motor(M_R) \wedge enc(E_L) \wedge enc(E_R)$$

What is missing to finalize the FOL model, is a representation of the structure. This can be easily done, connecting the ports of the components:

$$val(out(M_L), X) \leftrightarrow val(in(W_L), X) \wedge val(out(W_L), X) \leftrightarrow val(in(E_L), X)$$
$$val(out(M_R), X) \leftrightarrow val(in(W_R), X) \wedge val(out(W_R), X) \leftrightarrow val(in(E_R), X)$$

Although we do not model the control component directly, we add some rules stating what should be the case for given motion patterns like stop, going straight, or rotating. The following rules formalize the motion patterns. There we only state the expected values of the different ports for the motors so that they would provide the expected motion pattern, which should also be visible at the outputs of the encoders.

$$stop \rightarrow (val(in(M_L), 0) \wedge val(in(M_R, 0)))$$
$$straight \rightarrow (val(in(M_L), v_n^+) \wedge val(in(M_R, v_n^+)))$$
$$rotclk \rightarrow (val(in(M_L), v_n^+) \wedge val(in(M_R, v_n^-)))$$
$$rotinvclk \rightarrow (val(in(M_L), v_n^-) \wedge val(in(M_R, v_n^+)))$$

The set of all the discussed FOL rules constitute the system description for the robot example $SD_R$.

After formalizing a diagnosis system, we have to state a diagnosis problem. Diagnosis is necessary if an observed behavior is in contradiction with the expected behavior, which we derive from the system description, i.e., the model. Hence, a diagnosis problem has two ingredients: (1) the diagnosis system and (2) a set of observations stating values for connections between components or equally good the ports of components. Formally, a diagnosis problem is defined as follows:

**Definition 2 (Diagnosis Problem)** A tuple $(SD, COMP, OBS)$, where $(SD, COMP)$ is a diagnosis system and $OBS$ a set of observations, is a diagnosis problem.

For our robot example, we can easily state a diagnosis problem. For example, let us assume the faulty behavior we depict in Fig. 2. In case of the expected straight-line movement, we see a curve going to the left. Let us further assume that the encoder of the left wheel does not give us back the nominal value but the one of the right wheel does. Note that this assumption explains the actual behavior assuming that the left motor has less number of revolutions than expected. For this diagnosis problem, we can easily state the observations $OBS_R$:

$$OBS_R = \{straight, \neg val(out(E_L), v_n^+), val(out(E_R), v_n^+)\}$$

Note that predicates or rules given in a set are assumed to be true. Thus all elements of such sets can be considered as being connected using logic conjunctions, i.e., $\wedge$. Given a diagnosis problem, we are now interested in finding diagnoses. We first define a diagnosis formally.

**Definition 3 (Diagnosis)** Given a diagnosis problem $(SD, COMP, OBS)$. A set $\Delta \subseteq COMP$ is a diagnosis if and only if $SD \cup OBS \cup \{Ab(C)|C \in \Delta\} \cup \{\neg Ab(C)|C \in COMP \setminus \Delta\}$ is satisfiable.

In this definition of diagnosis, we are searching for an assignment of health states to all components, which eliminates all contradictions with the given observations. For example, $(SD_R, COMP_R, OBS_R)$ would lead to a contradicting logical sentence when assuming all components to be correct, i.e., setting their corresponding negated predicate $\neg Ab$ to true, because in this case we would expect $val(out(E_L), v_n^+)$ to be true, which contradicts $OBS_R$. When setting $Ab(M_L)$ to true and all other components to be working as expected, we are able to eliminate the contradiction. Hence, $\{M_L\}$ is a diagnosis. Unfortunately, assuming all components to be faulty would also be a diagnosis according to Definition 3. Therefore, we need a stronger definition of diagnosis, which focuses on parsimonious explanations.

**Definition 4 (Minimal Diagnosis )** Given a diagnosis problem $(SD, COMP, OBS)$. A diagnosis $\Delta$ for $(SD, COMP, OBS)$ is a minimal diagnosis if and only if there exists no diagnosis $\Delta'$ that is a subset of $\Delta$.

The definition of minimal diagnosis assures that only the smallest diagnoses in terms of the subset relation are considered. In case of our robot example, we obtain two minimal diagnoses $\{M_L\}$ and $\{E_L\}$ indicating that either the left motor or the left encoder is not working as expected. From here on, we assume if not otherwise stated that we are interested in minimal diagnoses only.

Computing minimal diagnoses is computationally expensive. In particular, the problem of searching for a minimal diagnosis is NP-complete providing that a theorem prover or any other reasoning algorithm can check satisfiability in linear time, which is not even the case for propositional logic where we know that the satisfiability problem (SAT) is itself NP-complete. However, in practice this is not so much a big deal, because we are mostly not interested in finding all minimal diagnoses but also the smallest ones with respect to cardinality. Moreover, in most cases there are enough single fault or double fault diagnoses, and the challenge is more to reduce the number of diagnosis candidates.

In literature there are many diagnosis algorithms discussed in detail. Greiner et al. [25] discussed a corrected version of Reiter's diagnosis algorithm outlined in [46], which makes use of conflicts for computing diagnoses. For more details about conflicts and further issues regarding MBD we refer to [46]. More recently, Felfernig et al. [18] presented the *FastDiag* algorithm, which allows computing diagnoses directly from the model. Nica et al. [41] presented an empirical evaluation of different diagnosis algorithm focusing on runtime. To be self-contained we briefly outline a simple algorithm for diagnosis, which makes use of a theorem prover like *Prover9* [35] for checking satisfiability of the system description, the observations together with the health assumptions. The theorem prover is called using the function **TP** and takes a set of rules and facts as input. It returns consistent if the given argument is satisfiable and inconsistent, otherwise.

The **CompMBD** algorithm (Algorithm 1) computes minimal diagnoses up to a cardinality $n$. Minimality is assured because in Line 4 we remove all supersets of components that are already diagnoses. The algorithm obviously computes all

**Algorithm 1 CompMBD** $((SD, COMP, OBS), n)$

**Input:** *Given a diagnosis system* $(SD, COMP)$, *a set of observations* $OBS$, *and a number* $n \in \{1, \ldots, |HYP|\}$.
**Output:** *A set of minimal abductive diagnoses up to cardinality n*

```
1:  Let Δ_S = ∅
2:  for i = 1 to n do
3:      Let C be the set of all combinations of elements of HYP of size i.
4:      Remove from C all elements where there exists a subset in Δ_S.
5:      for Δ in C do
6:          if TP(SD ∪ OBS ∪ {Ab(x)|x ∈ Δ} ∪ {¬Ab(x)|x ∈ COMP \ Δ}) is consistent
            then
7:              Add Δ to Δ_S.
8:          end if
9:      end for
10: end for
11: return Δ_S
```

diagnoses because we consider all combinations of components of a particular cardinality in each iteration. The elements in $\Delta_S$ have to be diagnoses according to Definition 3 because of the theorem prover call in Line 6. The runtime complexity is in the worst case $O(2^{|COMP|})$ ignoring the complexity of the theorem prover call because we search for diagnosis considering all combinations of components. **CompMBD** should therefore only be used in a small value of $n$ preferable smaller or equivalent to 3, which seems to be sufficient for practical applications, where we are mainly interested in a small number of diagnoses as already discussed previously.

But how to obtain a small number of diagnoses? How can we select the most likely diagnoses and do not need to consider all computed diagnosis candidates? In literature we find two practicable reasonable answers to these questions. De Kleer and Williams [13] introduced a probabilistic framework for MBD. There the authors discuss how to assign probabilities to diagnoses. For this purpose, we need the probability that a given component $C \in COMP$ fails, i.e., $p_F(C)$. If we know this probability for each component, we are able to assign a probability to each diagnosis $\Delta \subseteq COMP$ as follows:

$$p(\Delta) = \prod_{C \in \Delta} p_F(C) \prod_{C \in COMP \setminus \Delta} (1 - p_F(C)) \tag{5}$$

Hence, when using the probability of a diagnosis we can come up with a ranking of diagnoses presenting the most likely diagnosis first. Note that either the fault probabilities of components can be obtained due to the availability of reliable empirical data, e.g., from experiments, or we assign probabilities correspondingly to expectations. In the latter case, we might consider one component to be more likely to fail than another.

Alternatively to using probabilities, De Kleer and Williams suggested to make use of probing. For example, if we know that the encoder of our differential drive robot delivers the correct result given the motor's speed, we can conclude that only the motor has to be responsible for the wrong behavior. Hence, measuring values at certain connections between components can restrict the number of diagnoses. In their paper, De Kleer and Williams presented an optimal probing strategy that reduces the number of diagnoses with the least number of additional measurements. When using such a strategy, we are able to finally come up with a single fault diagnosis.

An alternative strategy for reducing the number of computed diagnoses is based on the introduction of fault models. There we make use of the faulty behavior of components to restrict the number of valid diagnoses given the observations. Struss and Dressler [57] discussed the integration of fault models into a general diagnostic framework, and—as already mentioned—De Kleer et al. [14] introduced a theory of diagnosis with fault models. Because of the increase in the search space, i.e., we do not only need to check all subsets of the set of components but also each possible mode, using fault models is not feasible forlarger systems. Friedrich et al. [22]

presented an alternative way of reducing the number of diagnoses without the need for fault models. The authors suggested to specify logical rules representing physical necessities. If such a rule is violated during diagnosis, there is contradiction and the corresponding behavior is physically impossible. Using physical impossibilities does not change the overall computational complexity and effectively reduces the number of computed diagnoses.

## *3.2 Abductive Diagnosis*

In contrast to MBD where we model only the correct behavior of components, abductive reasoning deals with models of the faulty behavior. From a logical perspective abductive reasoning makes use of hypotheses to explain certain symptoms. We have rules of the form *hypothesis* $\rightarrow$ *symptom*. Abduction is reasoning in the opposite direction of the implication $\rightarrow$. In case of abductive diagnosis the hypotheses represent different health states of components, which cause a certain behavior.

The following formalization is extension of the definition of Friedrich et al. [21] where the authors focused on propositional horn clause abduction problems, i.e., abductive diagnosis based on propositional logic dealing with implication rules and facts only. We first define abductive diagnosis systems similar to diagnosis systems in MBD.

**Definition 5 (Abductive Diagnosis System)** A pair $(SD, HYP)$ is an abductive diagnosis system where $SD$ is a logical model, and $HYP$ a finite set of hypotheses.

The definition of abductive diagnosis system is similar to Definition 1. The difference is that in case of MBD we have a set of components whereas in the case of abduction we use hypotheses. This change is due to the fact that in case of MBD we only consider the $Ab$ and $\neg Ab$ health state for each component but we may have more health states when dealing with abduction.

To illustrate the definitions, we make again use of our differential drive robot example but simplify the model. Instead of using all components of the robot (see Fig. 4), we focus on the motors only. We first specify what is going to happen in case the motor is running as expected, too slow, or too fast. Such a behavior can be formalized in a general way as follows:

$$\forall x : (motor(x) \wedge expected(x)) \rightarrow nominalspeed(x)$$

$$\forall x : (motor(x) \wedge tooslow(x)) \rightarrow reducedspeed(x)$$

$$\forall x : (motor(x) \wedge toofast(x)) \rightarrow increasedspeed(x)$$

In these rules the predicates $expected/1$, $tooslow/1$, and $toofast/1$ represent the health states of the motors. We state that our system has two motors:

$$motor(m_L) \wedge motor(m_R)$$

Furthermore, we introduce three rules stating that we can only have one speed for a motor at each time.

$$\forall x : \neg(nominalspeed(x) \wedge reducedspeed(x))$$
$$\forall x : \neg(nominalspeed(x) \wedge increasedspeed(x))$$
$$\forall x : \neg(increasedspeed(x) \wedge reducedspeed(x))$$

These three rules are for assuring that only one of the predicates representing speed can be true at a time. To finalize the model for abductive diagnosis, we introduce rules deriving a motion pattern for a mobile robot. The goal is to classify the observed behavior like given in Fig. 2 using a single predicate. Depending on the speed of the left and right motor, we obtain the following motion patterns:

$$(reducedspeed(m_L) \wedge nominalspeed(m_R)) \rightarrow leftcurve$$
$$(reducedspeed(m_L) \wedge increasedspeed(m_R)) \rightarrow leftcurve$$
$$(nominalspeed(m_L) \wedge increasedspeed(m_R)) \rightarrow leftcurve$$
$$(nominalspeed(m_L) \wedge reducedspeed(m_R)) \rightarrow rightcurve$$
$$(increasedspeed(m_L) \wedge reducedspeed(m_R)) \rightarrow rightcurve$$
$$(increasedspeed(m_L) \wedge nominalspeed(m_R)) \rightarrow rightcurve$$
$$(nominalspeed(m_L) \wedge nominalspeed(m_R)) \rightarrow straight$$

In this example, we distinguish three motion patterns, which we want to explain using abductive diagnosis. The introduced rules are element of $SD_A$ and $HYP_A = \{expected(m_L), tooslow(m_L), toofast(m_L), expected(m_R), tooslow(m_R), toofast(m_R)\}$. $(SD_A, HYP_A)$ states an abductive diagnosis system for our differential drive robot example.

Before defining abductive diagnosis formally, we introduce the definition of an abductive diagnosis problem.

**Definition 6 (Abductive Diagnosis Problem)** A tuple $(SD, HYP, OBS)$ is an abductive diagnosis problem where $(SD, HYP)$ is a abductive diagnosis system and $OBS$ is a set of observations.

This definition of a diagnosis problem is similar to the model-based diagnosis definition. For our running example the tuple $(SD_A, HYP_A, \{leftcurve\})$ represents an abductive diagnosis problem. A solution to this problem is an explanation for the observations, where an explanation is a conjunction of hypotheses that allow to derive the given observations.

**Definition 7 (Abductive Diagnosis)** Given an abductive diagnosis problem $(SD, HYP, OBS)$. A set $\Delta \subseteq HYP$ is a diagnosis if and only if

1. $SD \cup \Delta \models OBS$, and
2. $SD \cup \Delta$ is satisfiable, i.e., $SD \cup \Delta \not\models \bot$

In this definition, we make use of logic reasoning to define abductive diagnosis. In the first part, we require that the given observations, i.e., the observed symptoms, can be logically derived ($\models$) from the model $SD$ together with the hypotheses in $\Delta$. The second part assures that we cannot trivially obtain $OBS$, which would be the case if $SD$ together with $\Delta$ is inconsistent, because we can derive anything from inconsistent theories.

We furthermore define minimal abductive diagnoses as follows:

**Definition 8 (Minimal Abductive Diagnosis)** Given an abductive diagnosis problem $(SD, HYP, OBS)$. An abductive diagnosis $\Delta \subseteq HYP$ for the given diagnosis problem is a minimal diagnosis if and only if there is no other diagnosis $\Delta'$ that is a subset of $\Delta$.

For the differential drive robot example and the abductive diagnosis problem $(SD_A, HYP_A, \{leftcurve\})$ we are able to compute three different minimal abductive explanations:

$$\{tooslow(m_L), expected(m_R)\}$$
$$\{tooslow(m_L), toofast(m_R)\}$$
$$\{expected(m_L), toofast(m_R)\}$$

Either the left motor $m_L$ is too slow, providing that $m_R$ is running as expected or faster, or the right motor is running faster providing $m_L$ being slower or running at nominal speed. This result might be further reduced measuring the speed of the motors and comparing it with the expected values. It is worth noting that we are also able to make use of a slightly changed definition of diagnosis probability from Eq. (5). In case of abductive diagnosis we assume that we know the probabilities for each health state, i.e., each element in $HYP$. Hence, the definition of diagnosis probability can be simplified.

$$p_A(\Delta) = \prod_{x \in \Delta} p(x)$$

In case of our robot example, we might state that the expected behavior is much more likely. In this case we would prefer the first and the last of the three diagnoses.

Similar to MBD abductive diagnosis is at least NP-complete. In the following, we outline a basic algorithm for computing all abductive diagnosis up to a specific size. The algorithm is not optimized. For other algorithms we refer the interested reader to [39] and [31].

**Algorithm 2 CompAD** $((SD, HYP, OBS), n)$

**Input:** *Given an abductive diagnosis system* $(SD, HYP)$, *a set of observations* $OBS$, *and a number* $n \in \{1, \ldots, |COMP|\}$.
**Output:** *A set of minimal diagnoses up to cardinality n*

```
1:  Let Δ_S = ∅
2:  for i = 1 to n do
3:      Let C be the set of all combinations of elements of COMP of size i.
4:      Remove from C all elements where there exists a subset in Δ_S.
5:      for Δ in C do
6:          if TP(SD ∪ HYP) is consistent then
7:              if TP(SD ∪ HYP ∪ ¬OBS) is inconsistent then
8:                  Add Δ to Δ_S.
9:              end if
10:         end if
11:     end for
12: end for
13: return Δ_S
```

Algorithm 2 implements the abductive diagnosis algorithm **CompAD** where we go through all subsets of the hypothesis set and check for diagnosis. In Line 6 we first check whether the hypotheses are consistent with the system description. If this is true, we check whether we are able to derive $OBS$ using $SD$ together with $HYP$. This has to be the case if assuming that the observations are not valid ($\neg OBS$) together with $SD$ and $HYP$ leads to an inconsistency. The algorithm obviously terminates and computes all minimal abductive diagnoses up to a size $n$. Minimality is assured because we remove all supersets of already detected diagnoses in Line 4.

### *3.3 Summary on Model-Based Reasoning for Diagnosis*

The presented diagnosis approaches, i.e., MBD and abductive diagnosis, have been successfully used in practice. Whereas MBD makes use of a model capturing the correct behavior of components for computing diagnoses, abductive diagnosis relies on fault models. Both approaches rely on models that can be formally represented and where a reasoning mechanism is available for checking satisfiability. It is worth noting that we do not necessarily rely on FOL, propositional logic or any other logic formalism. We may also make use of constraints to represent models and constraint solving for checking satisfiability. For an introduction into constraints and constraint solving we refer to Rina Dechter's seminal book [16].

Although both diagnosis approaches are computationally demanding, current work, e.g., [41] and [31], has shown that MBD and abductive diagnosis can be used for practical applications and that their worst case computational complexity is not a limiting factor in practice. The only more severe restrictions of course are the necessity to have models that can be used for diagnosis. Obtaining such models

in practice is not always that simple, but due to the increasing importance of models for system development, this challenge seems to be solvable. There has been more recent work dealing with obtaining models from available development artifacts, see, e.g., [64].

## 4 Modeling for Diagnosis and Repair

Model-based reasoning requires that we have a model of the system we want to diagnose. In particular, we need a model that can be feed into a reasoning engine in order to determine consistency. In the previous sections we made use of FOL or other logics as underlying modeling language. However, model-based reasoning is not restricted to logic as a formalism for modeling. Beside the use of a certain modeling language a model to be used for model-based reasoning has also to provide means for setting or characterizing the health state of components or any other assumption we want to reason about.

In order to come up with models for diagnosis and repair, we first focus on some modeling principles that may be used. Wherever necessary, we distinguish modeling for MBD from abductive diagnosis. When starting modeling of systems the first part comprises coming up with the system's architecture, i.e., its parts and their interconnections together with the system's environment and interface. The interface of the system and its context is important in order to allow systems communicating with their environments. For modeling, we have to know this information as well. The architecture of a system can be seen as component-connection model where connections are interfaces between the ports of components. A connection is for exchanging information and data. From an abstract point of view each connection has a name and a type, e.g., a natural number or an array of reals.

After identifying the components, their interfaces, i.e., ports, and the connections, we have to represent the behavior of each component. In case of digital circuits such a behavior can be expressed using Boolean logic whereas in case of physical components differential equations would be a more appropriate form if we want to closely describe the real behavior of components over time. It is worth noting that the data types of the connections of course are also the same for connected components and should be used to come up with a component model. This component model has to capture the correct behavior in case of MDB and the fault behavior in case of abductive diagnosis. Before discussing the differences in modeling for MBD and abductive diagnosis, we first have a closer look at the underlying data types.

As already said, in principle formalisms used to describe models range from logic sentences to differential equations or even programs formulating a certain behavior. In case of diagnosis in general such fine-grained representations of reality are not needed and some form of abstraction is usually sufficient. Instead of using real valued connections we might be able to distinguish some finite

number of values that allow representing the behavior sufficiently. For example, let us consider again the robot example. Instead of dealing with the exact values of voltage and current used to control the number of rotations of a motor, it is sufficient to consider the case where there is voltage applied and thus the motor starts rotating, and the case where there is no voltage and the motor stops rotating. We may also want to distinguish a case where the voltage is too low or too high causing a decreased or increased speed. However, this depends on the current application.

For modeling the behavior we therefore require to search for the right degree of abstraction. Right in this context is informally speaking a set of values that can be distinguished leading to different behaviors. Using abstraction has the advantage of not requiring sophisticated simulation and requiring less computational resources. The use of abstraction for modeling is not new and has been proposed in the context of artificial intelligence almost 30 years ago. In qualitative reasoning (QR)[1] [62] researchers have been working on coming up with different kind of abstractions and also different underlying modeling principles. For the latter Kuiper's qualitative simulation [33], Forbus's qualitative process theory [20], and De Kleer et al.'s work on confluences [12] are worth mentioning. In qualitative simulation, ordinary differential equations are mapped to their abstract corresponding qualitative differential equations, which can be used to obtain all possible behaviors of a system without knowing the exact values. In addition, qualitative simulation also allows specifying new qualitative values in certain cases. Qualitative process theories make use of processes for specifying the abstract behavior of systems. There not components are of importance but processes determining the value of variables. There are still modeling environments, e.g., [4], available that are mainly used for formulating from biology and sustainable engineering.

In qualitative reasoning, we distinguish two possible cases of abstraction. Either there is a mapping of quantities directly to their corresponding qualitative values [10] or we represent deviations [56]. For example, in the case of an electrical motor we might be only interested in the case where the motor is stopping, rotating clockwise or anticlockwise. When knowing the electrical characteristics we may come up with the following mapping:

$$\begin{aligned} [-12\text{ V}, -0.5\text{ V}] &\mapsto -\omega \\ ]-0.5\text{ V}, 0.5\text{ V}[ &\mapsto 0 \\ [0.5\text{ V}, 12\text{ V}] &\mapsto +\omega \end{aligned}$$

[1] In qualitative reasoning variable values are abstract representations of their original domain. Instead of quantities like real numbers, qualitative representations are used for various purposes like simulation or diagnosis. Because of using qualitative values the name qualitative reasoning was established.

In this mapping, the assumption is that the motor can take voltages from $-12\,\mathrm{V}$ to $+12\,\mathrm{V}$ and that the rotational speed can be either anticlockwise ($-\omega$), zero (0), or clockwise ($+\omega$). The abstract domain $\{-\omega, 0, +\omega\}$ is totally ordered and we are also able to introduce abstract operations for such domains to be used further on for modeling the behavior. Of course because of abstraction we are losing information and sometimes the operators cannot distinguish potential outcomes. In this case, the question comes up to increase the abstract domain and to introduce new abstract values. This process can also be automated. Sachenbacher and Struss [48] presented a solution for automated domain abstraction.

In contrast to the direct representation of quantities as elements of an abstract domain, deviation models only consider as the name suggested deviations from nominal or expected behavior. Instead of stating that a value is 0.9 and therefore lower than the expected value of 1.0, the deviation, e.g., the value is small, is used. Hence, in case of deviation models we do not have a mapping of values to their qualitative representation but a mapping of deviations to their representation, e.g., "$<$" for stating a value to be smaller as expected. Deviation models have been used successfully in diagnosis, e.g., [56]. For more information about qualitative models have a look at [58, 59].

In the following, we discuss providing models for MBD and abductive diagnosis separately. Let us start with *modeling for MBD*. As already outlined we have a component-connection model and (possible abstract) data types for the connections, ports, and interfaces to the system environment and context. What we need now is the behavior of components. What we do first is to come up with certain types of components like a logical and with two inputs and one output. For each component type we need to specify a relationship between the different ports in case the component is working as expected. Hence, for each component $C$ of type $type$ we have to come up with rules of the form $\forall C : (type(C) \rightarrow (\neg Ab(C) \rightarrow Behav))$ where $Behav$ specifies the correct behavior of component $C$.

In case of our mobile robot example explained in Sect. 2 and later in Sect. 3, we defined the behavior of a motor as $(val(in(X), Y) \leftrightarrow val(out(X), Y))$ stating that every value $Y$ on the input port $in(X)$ has to be transferred to its output $out(X)$ and vice versa. Hence, we do not only specify one direction of data flow but formalize the model in a relational way ignoring the information whether a certain port works as input or output.

The behavior might also be given as equation like $v = R \cdot i$ representing a model for an electrical resistor. In such an equation we also do not have a data flow direction. We are only specifying relations that constraint given quantities or qualities. Of course it is necessary that the underlying reasoning mechanism is able to handle the given models. When relying on FOL we would not be able to specify the resistor model as given. In this case we may use a different kind of logic or a different (abstract) representation.

In addition, to the component models, we might also want to add rules stating physical impossibilities [22]. For example, let us consider an analog circuit comprising two bulbs in parallel coupled with a battery. A simple model would state that bulbs are lighting if there is voltage provided, and that a battery provides voltage.

In case one bulb is lighting but the other is not, there would be two diagnoses, i.e., the bulb that is not lighting and also the battery. The latter diagnosis is of course physically impossible, because an empty or broken battery cannot provide voltage. We can solve this issue via stating the impossibility: If a bulb is lighting, then there has to be a voltage. When adding such a rule, the battery cannot longer be a diagnosis.

*Modeling for abductive diagnosis* starts with the same input, i.e., the structure of the system. However, instead of defining the correct behavior, we are interested in the faulty cases and their consequences. Hence, we adopt a form of cause–effect reasoning, where causes are (faulty) health states of components and there effect are the symptoms we observe and want to explain. In this setting symptoms are deviations from the expected behavior. This type of modeling goes beyond MBD where we only have the health states correct or faulty, which we represent using $\neg Ab$ and $Ab$, respectively. In case of abductive diagnosis we have one or more health states for each component, which can also be represented using predicates. For each of these health states we present their consequences formally in the model. For this purpose, we would usually use rules of the form $\forall C : type(C) \rightarrow (cause_i(C) \rightarrow effects_i(C))$ where $type(C)$ is the type of the component, e.g., a Boolean and gate, $cause_i$ represents the health state, and $effects_i$ the consequences following from the given cause.

Such knowledge can be easily obtained from a failure mode and effect analysis (FMEA) [6, 26], which is regularly used in the context of safety critical systems in order to analyze the consequences of a fault and its corresponding risks. From the FMEA we obtain a table of the form:

| Failure mode | Effects | Risk |
|---|---|---|
| ⋮ | ⋮ | ⋮ |

This table can be almost directly mapped to cause–effect rules. For more details we refer to Wotawa [64] explaining the transformation in detail.

For practical applications it would be a significant advantage to have general modeling languages available that allows for writing models for model-based reasoning, like for simulation where we have beside Matlab/Simulink[2] other languages like Modelica [24]. Note that in contrast to simulation where all boundary conditions have to be known, in diagnosis we need models that allow to specify also the unknown behavior. Modeling languages for diagnosis have to deal with this specific requirement. Fleischanderl et al. [19] were one of the first introducing a modeling language for MBD considering the component-connection modeling principles and also different data types. The only limitation was the lack of considering time information in the models. Bonus et al. [5] presented an extension

[2]See https://de.mathworks.com/products/simulink.html.

of Modelica that can be used of modeling for MBD. Most recently, Nica and Wotawa [40] presented a language that also allows to come up with models considering time.

It is also worth mentioning modeling approaches that make use of physical models written in Modelica for fault localization using MBD and abductive diagnosis. De Kleer et al. [15, 36] discussed an approach for extracting diagnosis system models from Modelica models using explicit fault modes. Sterling et al. [55] presented an approach for mapping Modelica programs to diagnosis systems directly considering abstractions, and Peischl et al. [43] outlined the use of Modelica programs for obtaining cause–effect models for abductive diagnosis.

In summary, (1) modeling starts with identifying the boundaries of systems and their internal structure comprising components and their interconnections. In the second step (2) we have a look at the underlying data types of the connections and component ports as well as the interfaces of the system to its environment. For the data types, we have to elaborate on abstractions that are strong enough to distinguish the important behavioral aspects of components. If this is done, we formalize the component behavior in step (3) where we distinguish the case of MBD from abductive diagnosis. If we only know the correct behavior, we have to rely on MBD and define the component behavior as relations over the component ports. In case of abductive reasoning, we define rules of the form *causes imply effects* where a cause is a certain fault of a component. For abductive reasoning, it is also possible to determine the model from FMEAs, which are often used in engineering practice to determine the system's risk.

Until now, we have discussed the modeling steps for diagnosis. But are there any specific parts when dealing with *repair* or self-repair? Friedrich et al. [23] were one of the first considering repair as part of diagnosis. They formalized the repair process including diagnosis as one part. Basically, the general diagnostic process continues the diagnosis and probing loop until one candidate can be obtained, which should be replaced. Hence, in the simplest form repair is only a replacement of a faulty component. In case of fail-operational behavior or self-adaptive systems this would be close to the spare components that can be invoked whenever the component itself becomes faulty. However, as motivated in the introduction, we do not always have a spare component. Here we would require to change the system in a way such that the important functionality can still be guaranteed.

In the first case, i.e., using spare components for replacing broken components during runtime, we do not need an additional model. For this purpose, it is sufficient to know which component is faulty and should be replaced. Hence, both diagnosis approaches can be used directly. This is not possible, when wanting a system to behave as expected but maybe in a degraded mode. In such a case we need to model the degradation. We can do this via modeling which parameters have to be changed and how in case of a fault. Hence, we would need to specify also what can be done in case of a specific fault. In the next section, we discuss repair in much more detail and also provide examples for each of the different repair cases.

## 5 Self-adaptation Using Models

In this section, we discuss the use of model-based reasoning for self-adaptation. According to Weyns et al. [63] self-adaptation is the ability of a system to adapt to dynamic and changing operating conditions autonomously, i.e., without requiring human intervention. Self-adaptation can be seen as form of self-healing behavior, which is also known as autonomic computing where a system can detect, diagnose, and repair localized faults originating from software or hardware (see [30]). In the previous sections of this chapter, we already discussed the use of models for diagnosis purposes. Hence, we have to extend our framework to the capabilities of repair and in particular self-repair [9].

Repair itself can be classified according to [60] in (1) attributive repair and (2) functional repair. In attributive repair the idea is to restore the system to its original state, whereas in functional repair the focus is only on restoring the (important) functionality but not necessarily bringing the system back to its fault-free state. Hence, in functional repair we are also satisfied with a degraded behavior. Nevertheless in both types of repair we require a certain redundancy in order to bring back the system in a desired state. For example, in software-based self-repair [49], the reconfiguration is done on side of the user program to allow execution on the available processors.

It is worth noting that in order to achieve self-adaptation or self-healing the system itself has to monitor its health-state and react appropriately over time in case the actual behavior of the system is in contradiction with its expected behavior. After detecting such a relevant behavioral deviation, the system has to perform diagnosis using its underlying knowledge and later on self-repair. Hence, a self-healing system has to follow a certain system architecture. For example, IBM suggested such a reference model for autonomic control loops [28], which is also called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop. In this section, we introduce a simplified architecture, which requires monitoring, diagnosis (which is a kind of analysis), and repair (which comprises planning and execution). For all these three steps we rely on a model, i.e., knowledge of the system.

In Fig. 5 we see on the left the classic architecture of a system that interacts with its environment. For this purpose, we have a sensor and an actuator level for obtaining information from the environment and interacting with the environment. The system and their in particular the control component makes use of the measurements coming from the sensor level and its internal state to compute values for the actuators. For example, our differential drive robot may localize an object at a certain position via a laser range sensor or a computer vision system and computes the commands for the motors for moving to detected object. In case of a fault in any parts of the system, the robot will not work as expected anymore because it cannot detect the misbehavior and also not react accordingly.

This situation is different in case of the system architecture on the right side of Fig. 5. There we have an additional smart diagnostics component, which takes the information provided by the sensors together with the current state of the control

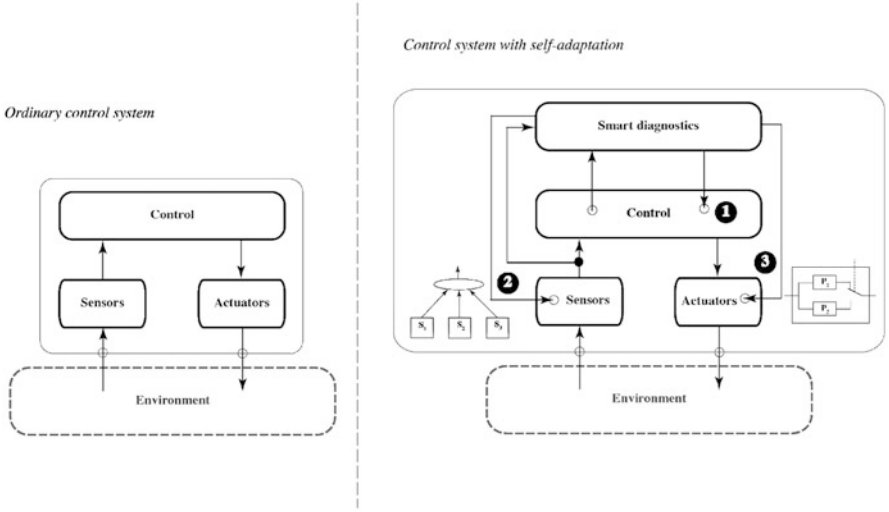


**Fig. 5** Control system without (left) and with (right) a smart diagnostic component

component to derive a diagnosis and afterwards repair actions, which might change the control component itself (1), the sensor information (2), or the actuators (3). For all three cases, we are going to outline examples later in this part of the chapter. Before, we discuss the structure of the smart diagnostics and how model-based reasoning can be integrated.

A smart diagnostics has to monitor the current system and based on this information to draw conclusions about the health state of the system over time. Hence, in every step at a certain point in time $t$, we have to evaluate the system's behavior with respect to deviations from the expected behavior. For this purpose, we can use the health state of the system obtained at the immediately previous time step $t - 1$ to predict a behavior. If this behavior is equivalent to the observed value, we know that the system is still in the same health state at time $t$ and no further action is required. Otherwise, we have to run diagnosis to explain the deviation and in case of identifying the root cause to repair the system.

It is worth noting that there might be the case that a fault occurring at a time step is only visible after more than one time step. In this case, diagnosis might give us back a wrong result. In order to overcome this issue, the time span between two time steps used for monitoring has to be defined as being large enough for all faults to be visible in the observations. If this is not possible, e.g., because of underlying physical processes requiring substantially different time spans for being observed, we have to introduce separated monitoring/diagnosis/repair cycles for all of these processes.

In Fig. 6 we illustrate time step $t$ using information from the previous step $t - 1$. Note that the observations are extracted from the sensor data and the internal state of the control block. When implementing the smart diagnostics there has to be

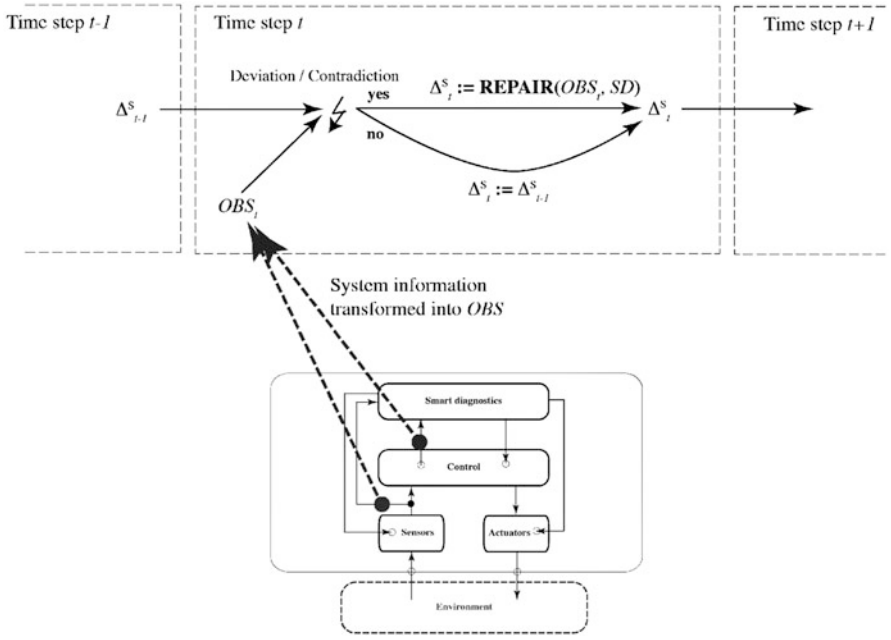


**Fig. 6** A monitoring/diagnosis/repair cycle at time step $t$

a program mapping the monitored variables to their corresponding observations $OBS_t$. This mapping does not necessarily need to set all potential observations. There may be observations that cannot be obtained at all points in time. But this is not a problem for model-based reasoning because there computation is done on available information only. It is also worth noting that in case of FOL as underlying modeling language, the mapping has to set the truth values of predicates. All observations that can be obtained must be true at the certain point in time $t$.

In Fig. 6 we also see that in case of a contradiction we call a method **REPAIR** and not diagnosis. This method calls a diagnosis algorithm for identifying candidates, selects the best candidate, and computes necessary repair steps either to bring the system into a correct state again or in a degraded mode where the system still can deliver its required functionality. Depending on the repair step the health state of a system $\Delta$ may or may not change. In case of using spare parts the system after repair should incorporate only healthy components. In a degraded mode, the system has some components still not working as expected but which have been compensated. Formally, we can define a system health state as mapping of components to their health states.

**Definition 9 (System Health State)** A system health state $\Delta^S$ for a system with a set of components $COMP$ is a sequence of pairs $\langle (c_1, h_1), \ldots, (c_{|COMP|}, h_{|COMP|}) \rangle$ where for all components $c \in COMP$ we have a corresponding (component) health state $h$.

**Algorithm 3 REPAIR** $(SD, OBS)$

**Input:** *Given the model of a SD and a set of observations OBS at the current time t*
**Output:** *A system health state* $\Delta^S$

1: Compute the set of diagnosis $\Delta_S$ using either **CompMBD** or **CompAd** depending on the diagnosis method used.
2: Select the one diagnosis $\Delta$ from $\Delta_S$.
3: Apply repair mechanisms using $\Delta$.
4: Let $\Delta^S$ be the old system health state at time $t-1$.
5: **if** Spare part repair **then**
6: Change the health states in $\Delta^S$ for all components in $\Delta$ to their correct state, i.e., $\neg Ab$.
7: **end if**
8: **if** Compensating action repair **then**
9: Change the health states in $\Delta^S$ for all components in $\Delta$ with their corresponding health state provided in $\Delta$.
10: **end if**
11: **return** $\Delta^S$

In a smart diagnostics the system health state is continuously adapted over time using the current observations and the past system health state. In the following, we first summarize the **REPAIR** algorithm and afterwards introduce three possible cases how such a smart diagnostics maybe react and why.

Algorithm 3 implements the necessary steps for repairing a system. The way a system is repaired is not explicitly stated in the algorithm and depends on the kind of possible action. However, we distinguish two cases either repairing via using new components or compensating actions. A compensating action in the context of this work is any change of parameters or health states of the system that still allow the system to behave as close as possible to its expectations. Compensating actions are very much application specific. Furthermore, in **REPAIR** we also do not specifically describe how to select a diagnosis. In practice, we either make use of probabilities for obtaining the most likely diagnosis, or we may make a random selection. Ideally, the diagnosis itself should only provide us with one diagnosis. However, that cannot be assured always. Moreover, if we select a wrong diagnosis, this will be visible later in time when there is again a deviation with the expected values. In such a situation we may consider changing the probabilities of faults and finally improve reasoning and selection over time. De Kleer and Williams [13] suggested such a process. The only issue that has to be assured is that the selection of the diagnosis does not violate certain properties like safety. If this can be the case we further are able to use such properties during diagnosis selection.

We now discuss the three cases of faults with corresponding repair that may occur during operation of a system. In Fig. 5, we see three different locations of a fault: ❶ indicates a fault in the control block, ❷ states a fault in the sensor block, and ❸ represents the case of a fault on side of the actuators. For all three cases, we discuss examples.

Control (❶): Let us consider the differential drive robot from Fig. 3 and its architecture depicted in Fig. 4. Instead of modeling the behavior using qualitative

domains, we now make use of equations and variables to specify the behavior. In particular for motors we say that the voltage applied is proportional to their number of rotations. The following rule states this behavior where $s_X$ represents the speed of the motor and $v_X$ the applied voltage. In this rule we set the constant stating the proportion between speed and voltage to 1.0.

$$\forall X : motor(X) \rightarrow (\neg Ab(X) \rightarrow (s_X = 1.0 \cdot v_X))$$

Encoders can be similarly modeled. Instead of speed a rotational encoder (or wheel encoder) returns pulses. In the following rule we assume that there are 36 pulses delivered for one full rotation. Hence, the number of pulses $t_X$ is a function $36 \cdots s_X$ where $s_X$ is the speed that should be measured.

$$\forall X : enc(X) \rightarrow (\neg Ab(X) \rightarrow (t_X = 36 \cdot s_X))$$

In addition, we have to model the structure, i.e., the components and connections, of the system:

$$motor(m_L) \wedge motor(m_R)$$

$$enc(e_L) \wedge enc(e_R)$$

$$s_{m_L} = s_{e_L} \wedge s_{m_R} = s_{e_R}$$

For this example, the above rules and equations represent the model $SD$ and $COMP = \{m_L, m_R, e_L, e_R\}$. Let us further assume that we have the following observations, where the voltage comes from the control block and the number of pulses from the encoders.

| $v_{m_L}$ | $v_{m_R}$ | $t_{e_L}$ | $t_{e_R}$ |
|---|---|---|---|
| 2.0 V | 2.0 V | 72 | 72 |

In this case, obviously everything is fine and there is no contradiction with the expectations. Let us now consider the case of a robot following the wrong trajectory depicted in Fig. 2. For this case, we would receive observations like the following, where the number of pulses on the left side is small than expected:

| $v_{m_L}$ | $v_{m_R}$ | $t_{e_L}$ | $t_{e_R}$ |
|---|---|---|---|
| 2.0 V | 2.0 V | 54 | 72 |

Using MBD, we are able to compute the two minimal diagnoses: $\{m_L\}$ and $\{e_L\}$. This result can be further improved when using physical impossibilities or other known properties, e.g., stating that a broken encoder would not deliver any pulses, i.e., $\forall X : enc(X) \rightarrow (\neg Ab(X) \vee t_X = 0)$. In addition, we might extend the model also to *compensate* the fault. In this example, the left motor is not

running the right speed. If extending the voltage, we may speed up the motor. Hence, we may come up with a component representing the proportionality factor between the voltage and the speed. The new model would look like:

$$\forall X : motor(X) \rightarrow (\neg Ab(X) \rightarrow (s_X = c_X \cdot v_X))$$
$$\forall X : const(X) \rightarrow (\neg Ab(X) \rightarrow (c_X = 1.0))$$
$$const(c_L) \wedge const(c_R)$$
$$c_{m_L} = c_{c_L} \wedge c_{m_R} = c_{c_R}$$

The set of components would be $COMP = \{m_L, m_R, e_L, e_R, c_L, c_R\}$. Using this model together with the rule stating the property that broken encoders do not deliver pulses, we obtain again two minimal diagnoses but this time: $\{m_L\}$ and $\{c_L\}$. The encoder is not a diagnosis anymore because we observe pulses. Let us now take a closer look at diagnosis $\{c_L\}$. When the constant is wrong, any value for variable $c_{c_L}$ can be inserted. Using an equation solver a value of 0.75 for $c_{c_L}$ explains the lower value of $t_{e_L}$. Assuming that this reduction of speed when providing the same voltage is—more or less—equally distributed over the working range of the input voltage, we would multiply the voltage with $\frac{1}{0.75}$ which equals to 1.33 before using it as input for the left motor.[3]

In this example, the compensation would be valid until new observations contradict again the behavior, maybe leading to further adaptations of the input voltage. Moreover, the diagnosis $\{c_L\}$ with its corresponding compensating value would be part of the system health state of the differential drive robot. The repair rule for this diagnosis would state how to compute the compensating value and how to apply it. The application may be either part of the component block, or before or in the actuator block of the smart diagnostics architecture (see Fig. 5 right picture).

Sensor (❷): In the second example, we illustrate how smart diagnostics can be used for sensor fusion and there in particular on identifying contradictions in sensor information and to react in an appropriate manner. For illustration purposes let us consider a mobile robot comprising the three sensors, inertial measurement unit (IMU), computer vision system (CVS), and the wheel encoders. All of this sensor can be used to identify the direction a robot is moving. From the IMU we would get an acceleration vector in the direction of movement when starting. The CVS can provide among other things an optical flow indicating the direction, and from the encoders we receive pulses where the difference indicates the direction. In the following we formalize the behavior of the different sensors:

[3] In practice, someone might make use of a more sophisticated process after detecting such a fault. A search procedure might be used to find the right voltage levels for the left motor in order to guarantee moving on a straight line.

$$\forall X : imu(X) \rightarrow (\neg Ab(X) \rightarrow (\forall Y : accdir(X, Y) \leftrightarrow dir_I(X, Y)))$$

$$\forall X : cvs(X) \rightarrow (\neg Ab(X) \rightarrow (\forall Y : optflowdir(X, Y) \leftrightarrow dir_V(X, Y)))$$

$$\forall X : encs(X) \rightarrow \left( \neg Ab(X) \rightarrow \left( \begin{array}{l} (equiv(X) \leftrightarrow dir_E(X, straight)) \wedge \\ (rightgreater(X) \leftrightarrow dir_E(X, left)) \wedge \\ (leftgreater(X) \leftrightarrow dir_E(X, right)) \wedge \end{array} \right) \right)$$

Note that $dir_I/2$, $dir_V/2$, and $dir_E/2$ are predicates for representing the current direction of the IMU, the CVS, and the encoders, respectively. In this example, we do not distinguish the encoders for the different wheels. We only assume that we know their number of pulses and are able to compare them leading to the respective truth values for the corresponding predicates.
In order to check consistency we further have to come up with consistency properties stating that all the different direction values should be the same.

$$\forall X : \forall Y : \forall Z : \forall V : (dir_I(X, V) \wedge dir_V(Y, V) \wedge dir_E(Z, V)).$$

Moreover, we have to state that each sensor is only allowed to deliver one value at a particular time.

$$\forall X : \forall V : \forall W : dir_I(X, V) \wedge dir_I(X, W) \rightarrow V = W$$

$$\forall X : \forall V : \forall W : dir_V(X, V) \wedge dir_V(X, W) \rightarrow V = W$$

$$\forall X : \forall V : \forall W : dir_E(X, V) \wedge dir_E(X, W) \rightarrow V = W$$

After specifying the structure of the system comprising three components $COMP = \{imu1, cvs1, encs1\}$ using the following rules, we obtain a complete model for sensor fusion:

$$imu(imu1) \wedge cvs(cvs1) \wedge encs(encs1)$$

Let us assume now that we obtain the following observations:

$$OBS = \{accdir(imu1, straight), optflowdir(cvs1, right), \\ equiv(encs1)\}.$$

From $SD$ and $OBS$ and assuming all components to work correctly we get a contradiction because the CVS is returning a different value. From the model, we are able to compute two minimal diagnoses, i.e., $\{cvs1\}$ and $\{imu1, encs1\}$. When focusing on the smallest diagnosis first, we would disable the CVS, which is the repair action for this example. Note that in this example we do not have a compensating action. Instead we simply turn off a component, which would not influence the rest of the system because the other two sensors determine the value used for further controlling the robot.

Actuator (❸): In the last example, we show the case of spare parts. Assume that we have the differential drive robot but this time we add power circuits for providing enough voltage and current for the motors. When using a similar model than in Sect. 3, we can determine which motor is faulty and therefore, which power circuit should be replaced. If we have a hardware architecture with integrated spare parts that can be enabled, the repair action would simply enable the new power circuit and disable the old one.

The presented approach for repairing detected and localized faults based on models has the advantage of relying on system models and therefore always delivering the best possible diagnoses for the given observations and the system model. We do not rely on data for learning diagnosis, therefore the approach can be used even in situations where such data is not available, e.g., after the development of systems where no feedback from its use is available. The approach allows for implementing fail-operational systems. We can assure that the proposed diagnosis fulfills given properties before applying it. Furthermore, the underlying algorithms deliver all minimal diagnoses as explained in this chapter. In order to prove that the approach is working as expected, we have to validate the model and there in particular their capabilities of specifying the behavior of the system. In this section, we further discussed how to use diagnosis for coming up with repair suggestions and how to integrate them in a smart diagnostics.

## 6 Related Research

The use of model-based reasoning for self-adaptation is not new. Rajan et al. [45] discussed the use of model-based reasoning in a combination with planning to form a new control system for an automated space probe, which was successfully tested in space. Later Hofbauer et al. [27] and Brandstötter et al. [3] introduced a model-based adaptive system that allows a robot adapting its drive in case of a fault in a motor. There the authors suggested to use hybrid automata for modeling where each state of the automata represents a certain health state of the robot. Besides diagnosis the approach also allowed to adapt the kinematics autonomously.

Besides reacting on hardware faults, there is also work on dealing with software faults occurring during operation. Steinbauer et al. [54] discussed a model-based approach for a mobile robot control system that is able to reduce the number of software processes that have to be restarted in case of a crash. The presented approach relies on an abstract model of the software considering the dependencies between software components. For an overview of the use of model-based reasoning to self-adaptive behavior, we refer the interested reader to [53].

Krenn and Wotawa [32] introduced another interesting approach to implement self-healing behavior based on models. There the model captures the ordinary behavior using rules that have to fulfill a certain goal, e.g., obtaining sensor measurements and sending them to a particular server. By specifying alternative

rules to achieve a certain sub-goal, it is possible to easily add redundancy. The execution system searches for those rules that can be executed at a particular point in time considering such alternative rules.

In the context of self-healing system research there is a lot of research dealing with self-adaptation and self-repair. Some like [9] discuss changes in traditional design practice in order to implement self-repair for producing systems comprising hardware and software components. For reconfiguration of electronic devices we refer, for example, to [2, 37, 47, 61]. For self-adapting software have a look at [38, 51] and more recently [44, 49, 52, 63]. Such software systems have to have the ability to modify itself in response to a change in its operating environment.

In the domain of automotive systems Seebach et al. [50] presented an approach for implementing self-healing behavior. There the authors made use of the adaptive cruise control (ACC) system as example application. In their approach the ACC system is deactivated and restarted after reconfiguration. Especially, in case of automotive systems fail-safe properties have to be fulfilled even after reconfiguration. Barbosa et al. [1] introduced the use of the formal modeling language Lotos to check monitored quantities over time. In particular, the presented tool monitors the execution traces generated by a self-adaptive system and annotates the probabilities of occurrence of each system action on their respective transition on the system model, created at design time as labelled transition system (LTS) that is used for checking properties.

In the context of cyber-physical systems there has also been research published. Niggemann and Lohweg [42] discussed the state of the art of diagnosis of cyber-physical systems and the research questions but focusing more on production systems. Mahadevan et al. [34] introduced the application of causal diagrams for fault localization. The approach presented in this work relies in contrast to these papers on model-based reasoning based on formal models.

## 7 Conclusions

In this chapter, we discussed the basic principles and foundations behind model-based reasoning in detail. We motivated why reasoning based on models is of particular interest for self-adaptive systems using a running example from the autonomous mobile robot domain. We illustrated different challenges that occur in this domain and discussed possible solutions. Regarding diagnosis we introduced two different methods, i.e., (1) model-based diagnosis and (2) abductive diagnosis. The first relies on models of system components only considering the correct behavior. The latter requires knowledge about faults and their corresponding behavior. For both approaches we outlined a simple algorithm allowing for computing minimal diagnoses from models and observations.

Because modeling is the important part, when introducing model-based reasoning, we also explained how to model for diagnosis. We outlined a process that starts with a system architecture comprising components and connections. From

this discussed how to come up with abstract models that map potentially infinite domains into a small set. For this purpose, we referred to qualitative reasoning and the abstractions discussed there. Furthermore, we introduced concepts for extracting models from other development artifacts like FMEAs or simulation models.

After presenting the basic concepts, we focused on the integration of diagnosis for self-adaptive systems. There we introduced a system combining a smart diagnostics with an ordinary control system interacting via sensors and actuators with its environment. We discussed the basic repair cycle and an algorithm for combining diagnosis with repair. In contrast to previous work we did not only rely on simple repair actions, e.g., replacing one component with its spare part, but also discussed how compensating actions can be integrated and used in the proposed setting. For the latter it is worth noting that the system model itself can be used to obtain sufficient information about the compensation.

The presented approach can be integrated into cyber-physical systems like autonomous vehicles in order to implement fail-operational behavior where compensating repair as well as automated repair using redundant hardware that can be enabled during operation is required.

**Acknowledgements** The research was supported by ECSEL JU under the project H2020 737469 AutoDrive—Advancing fail-aware, fail-safe, and fail-operational electronic components, systems, and architectures for fully automated driving to make future mobility safer, affordable, and end-user acceptable. AutoDrive is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information on https://iktderzukunft.at/en/ bmvit. The author wants to thank Dr. Iulia Nica for providing an initial survey on self-healing systems and application inspiring parts of the related research section.

## References

1. Barbosa, D.M., Lima, R.G.D.M., Maia, P.H.M., Costa, E.: Lotus@runtime: a tool for runtime monitoring and verification of self-adaptive systems. In: IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 24–30. IEEE, Buenos Aires (2017)
2. Boesen, M., Madsen, J.: eDNA: a bio-inspired reconfigurable hardware cell architecture supporting self-organisation and self-healing. In: Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware Systems, pp. 147–154. IEEE Computer Society Press, Moscone Convention Center, San Francisco (2009). https://doi.org/10.1109/AHS.2009.22
3. Brandstötter, M., Hofbaur, M., Steinbauer, G., Wotawa, F.: Model-based fault diagnosis and reconfiguration of robot drives. In: IEEE/RSJ International Conference on Intelligent Robots and System, pp. 1203–1209. IEEE, San Diego (2007)
4. Bredeweg, B., Bouwer, A., Jellema, J., Bertels, D., Linnebank, F., Liem, J.: Garp3—a new workbench for qualitative reasoning and modelling. In: Proceedings of the 20th International Workshop on Qualitative Reasoning (QR-06), pp. 21–28. Dartmouth College, Hanover (2006)
5. Bunus, P., Isaksson, O., Frey, B., Münker, B.: RODON—a model-based diagnosis approach for the DX diagnostic competition. In: Proceedings of the International Workshop on Principles of Diagnosis (DX) (2009)

6. Catelani, M., Ciani, L., Luongo, V.: The FMEDA approach to improve the safety assessment according to the IEC61508. Microelectron. Reliab. **50**, 1230–1235 (2010)
7. Console, L., Torasso, P.: Integrating models of correct behavior into abductive diagnosis. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 160–166. Pitman Publishing, Stockholm (1990)
8. Console, L., Dupré, D.T., Torasso, P.: On the relationship between abduction and deduction. J. Log. Comput. **1**(5), 661–690 (1991)
9. Coyle, E., Maguire, L., McGinnity, T.: Self-repair of embedded systems. In: Proceedings of the Engineering Applications of Artificial Intelligence, vol. 17, pp. 1–9 (2004). https://doi.org/10.1016/j.engappai.2003.11.009
10. Dague, P.: Qualitative reasoning: a survey of techniques and applications. AI Commun. **8**(3/4), 119–192 (1995)
11. Davis, R.: Diagnostic reasoning based on structure and behavior. Artif. Intell. **24**, 347–410 (1984)
12. de Kleer, J., Brown, J.S.: A qualitative physics based on confluences. Artif. Intell. **24**, 169–203 (1984)
13. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. Artif. Intell. **32**(1), 97–130 (1987)
14. de Kleer, J., Mackworth, A.K., Reiter, R.: Characterizing diagnoses and systems. Artif. Intell. **56**(2–3), 197–222 (1992)
15. de Kleer, J., Janssen, B., Bobrow, D.G., Kurtoglu, T., Saha, B., Moore, N.R., Sutharshana, S.: Fault augmented Modelica models. In: 24th International Workshop on Principles of Diagnosis (DX), pp. 71–78 (2013)
16. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
17. Dudek, G., Jenkin, M.: Computational Principles of Mobile Robotics, 2nd edn. Cambridge University Press, New York (2010)
18. Felfernig, A., Schubert, M., Zehentner, C.: An efficient diagnosis algorithm for inconsistent constraint sets. AI EDAM **26**(1), 53–62 (2012). https://doi.org/10.1017/S0890060411000011
19. Fleischanderl, G., Schreiner, H., Havelka, T., Stumptner, M., Wotawa, F.: DiKe—a model-based diagnosis kernel and its application. In: Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI), Vienna (2001)
20. Forbus, K.D.: Qualitative process theory. Artif. Intell. **24**, 85–168 (1984)
21. Friedrich, G., Gottlob, G., Nejdl, W.: Hypothesis classification, abductive diagnosis and therapy. In: Proceedings of the International Workshop on Expert Systems in Engineering. Lecture Notes in Artificial Intelligence, vol. 462. Springer, Vienna (1990)
22. Friedrich, G., Gottlob, G., Nejdl, W.: Physical impossibility instead of fault models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), Boston, pp. 331–336 (1990). Also appears in Readings in Model-Based Diagnosis (Morgan Kaufmann, 1992)
23. Friedrich, G., Gottlob, G., Nejdl, W.: Formalizing the repair process. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), pp. 709–713. Wiley, Chichester (1992). Also appeared in the Proceedings of the Second International Workshop on Principles of Diagnosis, Milano (1991)
24. Fritzson, P.: Object-Oriented Modeling and Simulation with Modelica 3.3—A Cyber-Physical Approach, 2nd edn. Wiley-IEEE Press, Piscataway (2014)
25. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in Reiter's theory of diagnosis. Artif. Intell. **41**(1), 79–88 (1989)
26. Hawkins, P.G., Woollons, D.J.: Failure modes and effects analysis of complex engineering systems using functional models. Artif. Intell. Eng. **12**, 375–397 (1998)
27. Hofbaur, M.W., Köb, J., Steinbauer, G., Wotawa, F.: Improving robustness of mobile robots using model-based reasoning. J. Intell. Robot. Syst. **48**(1), 37–54 (2007)
28. IBM: An architectural blueprint for autonomic computing (2003)
29. ISO/IEC/IEEE: Systems and software engineering—vocabulary. 24765:2010(E), pp. 1–418 (2010). https://doi.org/110.1109/IEEESTD.2010.5733835
30. Kephart, J., Chess, D.: The vision of autonomic computing. Comput. Mag. **36**(1), 41–52 (2003)

31. Koitz, R., Wotawa, F.: On the feasibility of abductive diagnosis for practical applications. In: 9th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (2015)
32. Krenn, W., Wotawa, F.: Intelligent, fault adaptive control of autonomous systems. In: Madrid, N.M., Seepold, R.E.D. (eds.) Intelligent Technical Systems. Lecture Notes in Electrical Engineering, vol. 38, pp. 175–188. Springer, Berlin (2009). https://doi.org/10.1007/978-1-4020-9823-9_13
33. Kuipers, B.: Qualitative simulation. Artif. Intell. **29**, 289–388 (1986)
34. Mahadevan, N., Dubey, A., Karsai, G., Srivastava, A., Liu, C.C.: Temporal causal diagrams for diagnosing failures in cyber-physical systems. In: Proceedings of the Annual Conference of the Prognostics and Health Management Society (PHM). PHM Society (2014). https://www.phmsociety.org/node/1439
35. McCune, W.: Prover9 and mace4 (2005–2010). http://www.cs.unm.edu/~mccune/prover9/
36. Minhas, R., de Kleer, J., Matei, I., Saha, B.: Using fault augmented modelica models for diagnostics. In: Proceedings of the 10th International Conference on Modelica, Lund, pp. 437–445 (2014)
37. Moreno, J., Madrenas, J., Faura, J., Canto, E., Cabestany, J., Insenser, J.: Feasible evolutionary and self-repairing hardware by means of the dynamic reconfiguration capabilities of the FIPSOC devices. In: Proceedings of the Evolvable Systems: From Biology to Hardware (ICES 1998), pp. 345–355. Springer, Berlin (1998)
38. Musliner, D., Goldman, R., Pelican, M., Krebsbach, K.: Self-adaptive software for hard real-time environments. Intell. Syst. **14**(4), 23–29 (1999)
39. Ng, H.T., Mooney, R.J.: An efficient first-order horn-clause abduction system based on the ATMS. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), pp. 494–499. MIT Press, Anaheim (1991)
40. Nica, I., Wotawa, F.: The SiMoL modeling language for simulation and (re-)configuration. In: Proceedings of the International Conference on Current Trends in Theory and Practice of Computer Science, vol. 7147, pp. 661–672. Springer, Berlin (2012)
41. Nica, I., Pill, I., Quaritsch, T., Wotawa, F.: The route to success—a performance comparison of diagnosis algorithms. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI), Beijing (2013)
42. Niggemann, O., Lohweg, V.: On the diagnosis of cyber-physical production systems: state-of-the-art and research agenda. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI), pp. 4119–4126. Association of the Advancement of Artificial Intelligence, Menlo Park (2015)
43. Peischl, B., Pill, I., Wotawa, F.: Abductive diagnosis based on modelica models. In: 27th International Workshop on Principles of Diagnosis (DX) (2016).
44. Pilgerstorfer, P., Pournaras, E.: Self-adaptive learning in decentralized combinatorial optimization—a design paradigm for sharing economies. In: 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 54–64. IEEE, Buenos Aires (2017)
45. Rajan, K., Bernard, D., Dorais, G., Gamble, E., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P., Rouquette, N., Smith, B., Taylor, W., Tung, Y.: Remote agent: an autonomous control system for the new millennium. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI), Berlin (2000)
46. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
47. Rincon, F., Teres, L.: Reconfigurable hardware systems. In: Proceedings of the International Conference on Semiconductor, New York, vol. 1, pp. 45–54 (1998)
48. Sachenbacher, M., Struss, P.: Task-dependent qualitative domain abstraction. Artif. Intell. **162**(1–2), 121–143 (2005). https://doi.org/10.1016/j.artint.2004.01.005
49. Schölzel, M., Koal, T., Müller, S., Scharoba, S., Röder, S., Vierhaus, H.T.: A comprehensive software-based self-test and self-repair method for statically scheduled superscalar processors. In: 17th Latin-American Test Symposium (LATS), Foz do Iguacu, pp. 33–38 (2016). https://doi.org/10.1109/LATW.2016.7483336

50. Seebach, H., Nafz, F., Holtmann, J., Meyer, J., Tichy, M., Reif, W., Schäfer, W.: Designing self-healing in automotive systems. In: Xie, B., Branke, J., Sadjadi, S.M., Zhang, D., Zhou, X. (eds.) Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC'10), pp. 47–61. Springer, Berlin (2010)
51. Seltzer, M., Small, C.: Self-monitoring and self-adapting operating systems. In: The Sixth Workshop on Hot Topics in Operating Systems, California, pp. 124–129 (1997)
52. Shevtsov, S., Weyns, D., Maggio, M.: Handling new and changing requirements with guarantees in self-adaptive systems using SimCA. In: 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 12–23. Buenos Aires (2017)
53. Steinbauer, G., Wotawa, F.: Model-based reasoning for self-adaptive systems—theory and practice. In: Assurances for Self-Adaptive Systems: Principles, Models, and Techniques, pp. 187–213. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-36249-1_7
54. Steinbauer, G., Mörth, M., Wotawa, F.: Real-time diagnosis and repair of faults of robot control software. In: RoboCup. Lecture Notes in Computer Science, vol. 4020, pp. 13–23. Springer, Berlin (2005)
55. Sterling, R., Struss, P., Febres, J., Sabir, U., Keane, M.M.: From modelica models to fault diagnosis in air handling units. In: Proceedings of the 10th International Conference on Modelica. Linköping University Press, Lund (2014)
56. Struss, P.: Deviation models revisited. In: Working Papers of the 15th International Workshop on Principles of Diagnosis (DX-04) (2004)
57. Struss, P., Dressler, O.: Physical negation—Integrating fault models into the general diagnostic engine. In: Proceedings 11th International Joint Conference on Artificial Intelligence, Detroit, pp. 1318–1323 (1989)
58. Travé-Massuyès, L., Ironi, L., Dague, P.: Mathematical foundations of qualitative reasoning. AI Mag. **24**(4), 91–106 (2004)
59. Travé-Massuyès, L., Prats, F., Sánchez, M., Agell, N.: Relative and absolute order-of-magnitude models unified. Ann. Math. Artif. Intell. **45**(3–4), 323–341 (2005)
60. Umeda, Y., Tetsuo, T., Hiroyuki, Y.: A design methodology for self-maintenance machines. J. Mech. Des. **117**, 41–53 (1995)
61. Villasenor, J., Hutchings, B.: The flexibility of configurable computing. Signal Process. Mag. **15**(5), 67–84 (1998)
62. Weld, D., de Kleer, J. (eds.): Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann, San Mateo (1989)
63. Weyns, D., Holvoet, T.: An architectural strategy for self-adapting systems. In: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '07). IEEE Computer Society, Washington (2007). http://dx.doi.org/10.1109/SEAMS.2007.3
64. Wotawa, F.: Failure mode and effect analysis for abductive diagnosis. In: Proceedings of the International Workshop on Defeasible and Ampliative Reasoning (DARe) (2014)