Handbook of Knowledge Representation
Edited by F. van Harmelen, V. Lifschitz and B. Porter

DOI: 10.1016/S1574-6526(07)03010-6

Chapter 10

Model-based Problem Solving

Peter Struss

10.1 Introduction

The development of the concept of model-based systems was an answer to the limitations of rule-based "expert systems", which base problem solving (e.g., diagnosis) on a representation of experiential knowledge in a domain. These limitations are not due to the syntactic form of representing knowledge (rules), but result from the nature of the represented knowledge: termed "empirical associations" in the pioneering paper [11] or "shallow knowledge" in others. This has to be contrasted with "1st principles" knowledge (or "deep knowledge"), such as the representation of the understanding of the physical behavior of the components of a system.

To illustrate this distinction and its implications by an example, consider the simplified electrical subsystem of a vehicle comprising the starter, the rear lights, and the head lights with their power supply (Fig. 10.1(a)). Some simple diagnostic rules for such a system, gained from experience or some analysis of the system, might be

```
IF Engine_Does_Not_Start
        THEN Possible_Cause_Battery_Flat
IF Engine_Does_Not_Start
        THEN Possible_Cause_Starter_Defect
. . .
IF Rlights_On OR Hlights_On
        THEN NOT(Possible_Cause_Battery_Flat)
```

which would allow to suspect the starter, but not the battery, if the engine does not start and the lights are on. However, they lead to wrong consequences, when we face a system with two batteries, as indicated in Fig. 10.1(b).

Experience is obtained in a specific context. In our example, the specific structure of the system is compiled into the rules, it is implicit, and this is why the applicability of the last rule is limited to systems sharing the same structure or, rather, the same structural properties that underlie the rule. A rule may be reusable for the modified system (such as the first one), but the conditions for its reuse remain hidden. Furthermore, there is the question whether the empirical basis has the required coverage.

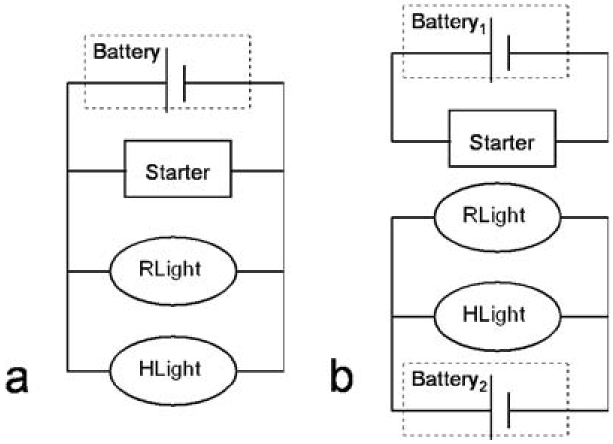


Figure 10.1: Two variants of electrical systems in a vehicle.

Even for moderately complex systems, we cannot expect that all possible faults have already been encountered in practice, let alone all combinations of independent faults.

More fundamentally, there may be no empirical data at all available for a particular kind of system. If we buy the latest model of a car, we would not accept the recommendation of a workshop mechanic that we should return with our problem next year to give them some time to gain experience. For certain systems and failures, we would not want to collect the empirical associations—think of airplanes or nuclear power plants.

It is a constitutive feature of human intelligence to extract principled knowledge from experience that can be used in a different context and for other purposes, and reproducing this capability is a major challenge to AI.

Taking a second look at the example, we notice that the rules do not only have a particular context in terms of the system structure compiled into them, they also represent the application of some principled knowledge to a **specific task**, namely diagnosis. However, the same fragment of knowledge, such as "A flat battery does not provide voltage and, hence, may cause the starter not to work", can also be used to solve a different task, such as failure-modes-and-effects analysis (FMEA), which aims at predicting the effect component failures have on the system function, the generation of a test that can reveal the presence of this fault, etc. In reflection of these challenges, model-based systems aim at

- representing the knowledge about a class of real-world systems as a **library of models** with a maximum of versatility and re-use to different system instances and for different tasks,
- providing model-based **problem solving engines** that support or automate the exploitation of such models to solve certain tasks.

These objectives meet urgent needs in industry, where complexity and variability of products demand computer support to capturing and applying the corporate knowledge. Also society benefits from powerful model-based systems, e.g., in improving the understanding, monitoring, and influencing of ecological, environmental, and climate systems.

These objectives strongly suggest the architectural principle of knowledge-based systems, namely a clear separation and independence of

- the **domain-specific knowledge** as a model-library, a declarative, decompositional representation of the behavior of elementary constituents of systems in the domain,
- the **task-specific knowledge**, in terms of problem solving engines that perform inferences based on a model library.

Independence of these two constituents of model-based systems is not to be understood at a low technical level (data structures), but at a conceptual level: the models should be stated in a way that is not committed to one particular task; and the problem solving engine should avoid encoding specificities of a particular domain and, hence, be able to operate on different model libraries. This is the basis for high reusability of both types of modules.

Of course, in practice (in research as well as in application-oriented work) the space of answers to the challenge has many dimensions. Perhaps more than in other areas of knowledge representation, the diversity of real-world problems induces a tremendous diversity in the proposed solutions. In this field, we are (or should be) facing systems in the real world, and there are many different kinds: electrical circuits, thermodynamic systems, water treatment plants, interacting species of flora and fauna, software, . . . We would like to solve tasks like system design, diagnosis, testing, repair, automated recovery, . . .

The Cartesian product *systems* × *tasks* is further expanded when researchers and developers choose formalisms (ordinary differential equations, finite state machines, predicate calculus, Bond graphs, Petri-nets, . . .) and apply their favorite inference scheme (qualitative simulation, finite constraint satisfaction, theorem proving, optimization, model checking, PROLOG, . . .). Although some modeling approaches seem to be more appropriate for certain classes of systems than others, the mapping *systems* $\leftrightarrow$ *models* is $m : n$, and so is the mapping *tasks* $\leftrightarrow$ *inference engines*.

As a result, any attempt of a comprehensive survey is prohibitive, even when confined to the solution ideas, let alone implementation. However, we will try to show that, at a certain level of abstraction, several tasks can be formalized using a small set of inferences (which can be realized in different ways). This will be done in the following section.

And we will choose a very general notion of “model” (which can be represented in many specific ways) and discuss required or advantageous properties (Section 10.3).

The remainder of the chapter will then be structured along different tasks. Diagnostic theories and systems (Section 10.4) will take the largest share for two reasons: diagnosis is the task with the most advanced theories and applications. On the other hand, some of the theories and implementation principles carry over to other problems as motivated in Section 10.2. We first present the foundations for a large class of diagnostic systems, consistency-based diagnosis based on component-oriented models, but will also identify its underlying assumptions and limitations and characterize alternative approaches.

Then we discuss test generation and diagnosability analysis (Section 10.5), generation of remedies (Section 10.6), and some other tasks (Section 10.7), and, finally, try to identify some major challenges in the field.

As stated before, due to the diversity of the solutions and the purpose and restriction of this chapter, our goal cannot be a comprehensive presentation of all proposed approaches and systems (and not even a comprehensive list of references), but, rather, conveying the key ideas of selected solutions with some formalization and, perhaps, some hints on a possible implementation. In the selection, we give preference to solutions that address the important requirements of the application context in a principled and general way over approaches that are heavily influenced by specific features of a particular application domain or that fail to reflect essential conditions of the real-world task.

10.2 Tasks

In this section, we characterize the essence of different tasks we would like to address based on some model. For this purpose, we are not very specific about the content of the model and the special form it is represented in. Requirements on the model, part of which follow from this analysis, will be discussed in the next section. Here, a model is a description of the possible ways a certain system can behave. This can be a real physical system or a hypothetical one (e.g., in design), a system that is in order or faulted (e.g., in diagnosis). In this section, we assume for the sake of a formal presentation that such a model, whatever the chosen representation is, can be equivalently stated as a set of logical formulas. Of course, in practice, representations will be chosen that are more suited for the description of physical systems. In this case, it has to be analyzed how the logical concepts and inferences carry over to the different formalism.

As it turns out, all we expect from a model is that it can be decided whether or not a certain behavior description contradicts the model (i.e. the notion of **consistency**) and whether it follows from the model (**entailment**).

10.2.1 Situation Assessment/Diagnosis

Diagnosis is about finding out that and why something does not behave the way it should. We have a model $MODEL_{OK}$ of the correctly working system, a set OBS of observations of the actual behavior of the system, and a set $GOALS$ specifying its intended behavior. Then, **fault detection**, the first step in diagnosis, means to check whether the joined theory is consistent

$$MODEL_{OK} \cup OBS \cup GOALS \not\models \bot$$

or, stronger, to ask whether the $GOALS$ are entailed:

$$MODEL_{OK} \cup OBS \models GOALS$$

We may assume that the system is well-designed, i.e., if nothing is broken, the specified behavior is guaranteed to be achieved,

$$MODEL_{OK} \models GOALS$$

In this case, the check is reduced to

$$MODEL_{OK} \cup OBS \not\models \bot$$

If this check reveals an inconsistency, we can conclude that $MODEL_{OK}$ does not describe the system under its current physical conditions; there must be a fault. In order to fix the problem, we need to know where the fault lies (**fault localization**) and/or what kind of fault is present (**fault identification**). In model-based diagnosis, this can be stated as the task of deriving a model $MODEL_F$ (or several alternative ones) that is, at least, consistent with the observations (*consistency-based diagnosis*)

$$MODEL_F \cup OBS \not\models \perp$$

or even entails them (*abductive diagnosis*, see Section 10.4.3).

In diagnosis, the space of models that are candidates for $MODEL_F$, is not arbitrary. Usually, the system performed well before and is now suffering from some particular malfunctions or disturbances. For instance, unless a major accident has happened, there will usually be one or two broken components in our car. This is why we can expect some restricted space of models that contains the solutions we are looking for, although it will often be too large to allow for an exhaustive consistency check of all candidates, and, hence, require search. In this search, we can exploit an ordering on the candidate models that is induced by the degree of deviations from $MODEL_{OK}$, e.g., indicated by the number of faulty components. This provides the basis for a hypothesize-and-test cycle where the new hypotheses are obtained by some elementary revision of the failing candidates, e.g., by assuming a different fault, an additional faulty component, etc. What we need in order to realize such a search-based approach is a module that checks the consistency of the model with the given observations and a component that produces new model hypotheses by revision of inconsistent ones based on some description of the possible disturbances in the model library (Fig. 10.2(a)). In Section 10.4, we present more details about such solutions.

10.2.2 Test Generation, Measurement Proposal, Diagnosability Analysis

If diagnosis does not provide a sufficiently focused answer, more observations are required to help discriminating between the remaining fault hypotheses. This means, we are looking for some stimulus *INP* to the system such that its observed response reveals the differences between the various hypotheses. In our model-based context, this means: given two behavior models $MODEL_1$ and $MODEL_2$, the target situation is

$$INP \cup MODEL_1 \models OBS_1$$

$$INP \cup MODEL_2 \models OBS_2$$

$$OBS_1 \cup OBS_2 \models \perp$$

Test generation is the task of determining inputs *INP* with this property and the appropriate observables, in case they can vary. In testing for diagnosis, this needs to be done for all pairs of models that represent relevant diagnoses. In end-of-line testing, $MODEL_{OK}$ needs to be discriminated from the models of relevant faults. **Measurement proposal** can be seen as a special case, where *INP* is fixed by the current situation, and the task is focused on determining where to probe for discrimination.

Also in the design phase of a system, this analysis can be relevant. **Detectability analysis** has to determine whether, under a given set of observables (e.g., by the sen-

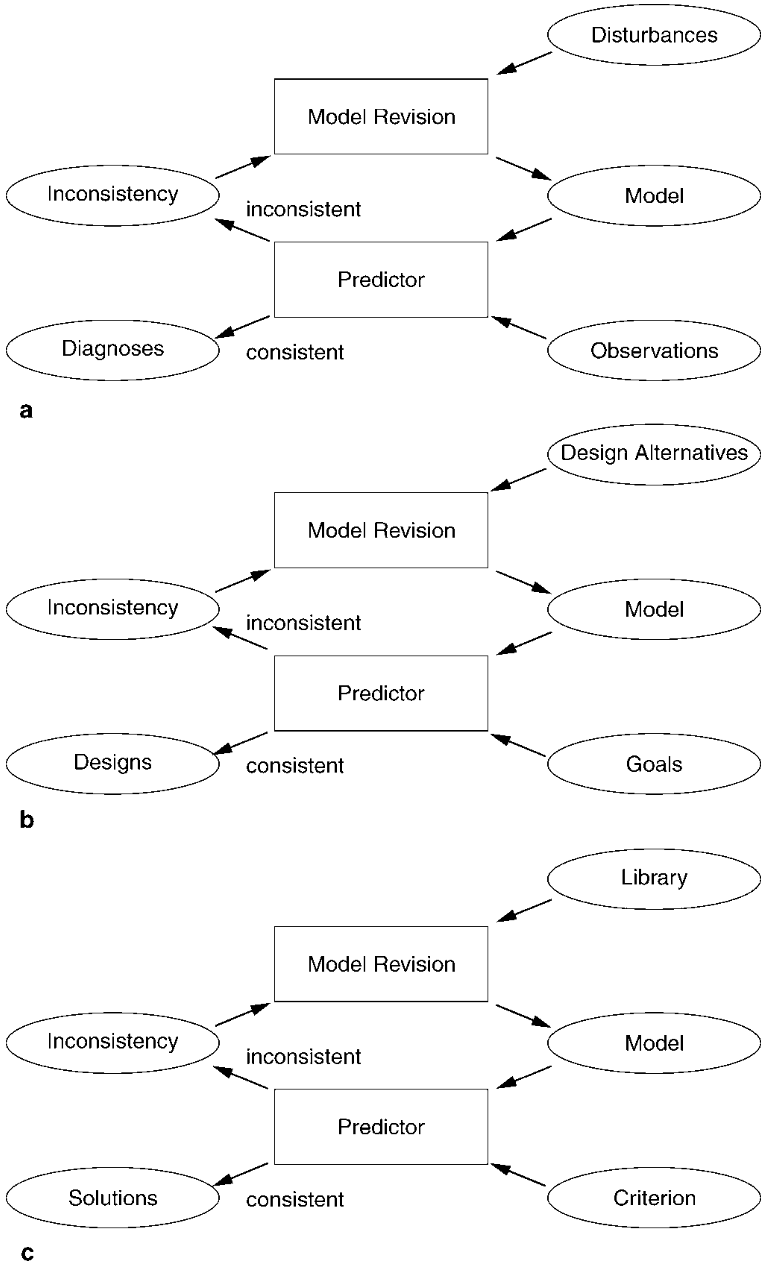


Figure 10.2: Consistency check and model revision in searching for diagnostic hypotheses (a), design solutions (b), and remedies (c).

sors built in), they reveal the distinction between a fault and the normal behavior, and, perhaps, under which conditions (represented by *INP*) this is the case, while **discriminability analysis** asks for distinguishing between two faults, which can be important to trigger some appropriate automatic response to the fault (e.g., in on-board recovery actions on vehicles). This analysis is relevant to sensor placement and also part of failure-modes-and effects analysis.

10.2.3 Design and Failure-Modes-and-Effects Analysis

The **design** of a system that has to generate a specified behavior demands strongly for a model-based approach, if a trial-and-error process by building physical prototypes should be avoided or limited. Unfortunately, in the general case, it is more challenging than diagnosis. If *GOALS* represents the behavior specification, then the design task can be solved by finding a model that entails this behavior:

$$MODEL \models GOALS$$

A necessary precondition for such a solution is that it is consistent with the specified behavior, which may be exploited at least in a first step to reduce the space of candidate models:

$$MODEL \cup GOALS \not\models \bot$$

We would usually not be satisfied by a product design that **may**, but is not guaranteed to, serve its purpose. However, the consistency check may be the only possible one in early phases of the design process, which may leave open the choice of specific components or parameters or even some structural properties, and it is helpful because it can refute certain design alternatives. Furthermore, the design process is rarely a single jump to a solution, but approaching one by modifying previously refuted design hypotheses in a way that inconsistencies with the specification are removed. Again, we can organize this process as a search in a space of candidate models (Fig. 10.2(b)). They need to be checked for consistency with the goals and, in case of inconsistency with the goals, revised by changing design decisions. What makes the task harder is, first of all, the nature and size of the space of possible alternatives. Usually, this space is much less restricted than, say, in diagnosis, where the structure of the system may be fixed and the possible component failures limited. In design, the structure may be what needs to be developed and modified.

Obviously, the revision-based search can only work if there is an initial hypothesis that can lead to a solution after a limited number of modifications. In fact, the vast majority in industrial design is not completely innovative, but emerges from a modification of a predecessor product. And often, the structure is more or less fixed, which turns design into the more tractable task of selecting appropriate components from a given set (**configuration**) or only determining parameters of fixed component classes (**parametric design**).

As a special task during design, **failure-modes-and-effects analysis**, has gained importance (and is mandatory, for instance, in the aeronautics and automotive industries). It is concerned with the task of making sure that, for a given design, even under the occurrence of a fault (usually a single fault) the resulting behavior of the system would not be critical or even catastrophic. The analysis has to find out for a set of given scenarios (e.g., the landing phase of an aircraft) and a set of relevant component failures whether one of the specified effects, i.e. violations of the functionality (e.g., "landing gear not extended"), can occur. The result of this analysis may be requested changes of the design.

Given a model of the system behavior under the presence of a possible failure, $MODEL_F$, it has to be determined whether it entails some *EFFECT* in a scenario specified by *INP*:

$$INP \cup MODEL_F \models EFFECT$$

or does not exclude (is consistent with) the effect:

$$INP \cup MODEL_F \cup EFFECT \not\models \perp$$

10.2.4 Proposal of Remedial Actions (Repair, Reconfiguration, Recovery, Therapy)

Diagnosis is only a step towards the real goal, which is restoring the functionality of a disturbed system, as far as it is possible. This is a trivial step, at least at the level of inferences, if it amounts to the **replacement** of broken components, in which case fault localization provides the direct answer. In other cases, built-in structural redundancy can be exploited for **reconfiguration** of a system in a way that (a part of) the objectives can be achieved despite a fault. For instance, breakers in a power network are opened and closed to provide continued power supply before the actual cause of a disturbance has been removed. This means to determine a target state, *STATE*, such that

$$STATE \cup MODEL_F \models GOALS$$

or, in the consistency-based form,

$$STATE \cup MODEL_F \cup GOALS \not\models \perp$$

Reconfiguration leaves the designed structure of the system unchanged and has a well-specified, though potentially large, search space: the space of states of the switching elements. The search can be guided by the number or cost of the required state changes with respect to the actual states.

In the more general case, which we may call **therapy**, remedial actions may have to modify the real system in order to bring it back to a healthy state. This often holds, for instance, for natural systems or plants that involve chemical or biological processes. Adding substances may trigger new processes and, hence, lead to a new system model:

$$ACTIONS \cup MODEL_F \models GOALS'$$

or, in the consistency-based form,

$$ACTIONS \cup MODEL_F \cup GOALS' \not\models \perp$$

$GOALS'$ will usually be some intermediate goals, which represent the direction towards the ultimate *GOALS*. Increased irrigation of the almost destroyed Everglades will only after some time lead to a healthy state of the flora and fauna, if at all.

We derive the same pattern again (Fig. 10.2(c)), and the feasibility will heavily depend on the size and structure of the space of revisions, which in this case correspond to the available remedial actions.

10.2.5 Ingredients of Model-based Problem Solving

This attempt to analyze and formalize the core of various real-world tasks and the exploitation of behavior models at a very abstract level reveals some of the fundamental technical tasks that have to be addressed by any model-based solution that aims at automating the respective problem solving. It also shows that they are shared across the various tasks, which opens the chance to reuse even algorithms, although the specific nature of the models and the structure of the model space will influence the details and appropriate heuristics.

This analysis, despite its abstract nature, also leads to some fairly important requirements on the modeling formalism which will be discussed in the next section.

10.3 Requirements on Modeling

In the previous section, we formalized the considered tasks using notions of consistency and entailment. This has sometimes led to the misconception that the system model has to be formulated as a logical theory (and has turned away some researchers, engineers, and users from this approach). While logic is one formalism with a precise semantics of entailment and consistency, it is not the only one, and, in fact, it is not an appropriate modeling language for most applications of model-based reasoning. Many applications lie in the engineering domain, others in social, ecological, biological, etc. domains and are difficult or impossible to model in first-order logic. Fortunately, this is not necessary. Although some widely used modeling formalisms can be translated into first-order logic, such as component-oriented modeling with finite domain constraints, even this is not a prerequisite for applying the problem solving engines we will discuss in the subsequent sections. This is possible thanks to the architectural principle of model-based systems, namely the separation of the model from the problem solving reasoning. The latter is often described in terms of logical inferences (although some of the most important and successful systems are not) which allows to analyze and prove properties of algorithms used in solutions, whereas the model is almost never stated in logic.

Of course, the modeling formalism has to fulfill certain theoretical and technical requirements in order to support the kind of problem solving described in the previous section, and we will now discuss these general requirements, rather than listing and describing candidates for modeling formalisms (algebraic and differential equations, qualitative differential equations, constraints, difference equations, causal graphs, rules, logic, finite state machines, Petri nets, discrete event models, Bond graphs, ...). This may seem to be a drawback, but it should be considered as an advantage, because this perspective allows for the exploitation of ideas, methods, and algorithms in combination with different types of models and for the choice of the models best suited for a particular domain and problem.

There are some fundamental requirements that originate from the application context and imply some of the technical ones.

- **Domain-oriented models**: this includes the **expressiveness** of models and the **efficiency** of model-based inferences, and, often, a trade-off between these two aspects. In contrast to a resistive circuit, a copier needs some representation of duration (of processing and transportation). A diagnosis system on-board a vehicle needs real-time performance. Model-based failure-modes-and effects analysis demands for qualitative models, since it aims at determining effects of classes of faults with unspecified parameters.

  In most areas, model-based reasoning meets a set of developed and established modeling formalisms and tools used in current practice. On the one hand, they promise to capture some of the essential features and, hence, cannot and should not be ignored by model-based systems. On the other hand, they often fail to provide some of the required capabilities that can be provided by AI techniques. Integration is often difficult, but important in order to obtain acceptance of the domain experts and users. If AI researcher ignore these aspects, this renders their work ineffective.

- **Libraries of reusable models**: model-based reasoning techniques rarely refer to a task that is not already performed by humans, and, often, performed quite well without an explicit representation of models. Model-based systems are only interesting if they offer some improvement in this performance, in terms of the quality of the result, or in terms of the cost needed to obtain the result. In any case, if the construction of the required model consumes more time than the traditional way of solving the problem, a model-based solution is not a solution. The fact that model-based reasoning aims at capturing the basic domain knowledge, which can be applied to different tasks and/or systems sets the challenge to represent this knowledge as a set of re-usable model fragments. This forms the basis for producing system models by composition of such model fragments, thus reducing the modeling efforts and time. Again, approaches that ignore this requirement, in treating a system model as a hand manufactured unstructured system model, fail to provide a suitable basis for solutions.

Together with these requirements, the formalized tasks presented in Section 10.2 translate into a set of relevant theoretical and technical properties and requirements of modeling.

10.3.1 Behavior Prediction and Consistency Check

Whatever the preferred modeling formalism is, in order to be useful for consistency-based problem solving, it has to have at least some sort of concept of consistency and, for abductive solutions, of entailment. Given some (fraction of) a model of a system's behavior, it must be possible to tell whether or not it contradicts given observations, goals etc. (and to draw conclusions about unobserved features, e.g., related to goals). This is a basic requirement and one that should be met by most modeling formalisms, because they are designed for prediction, and one can compare the predicted behavior to the observed or intended features. Nevertheless, in designing a model-based reasoner, it is important to precisely define the notion of inconsistency specific to a particular model-based predictor. If it can decide that a model is inconsistent (and, perhaps, which part of the model caused the inconsistency), this suffices to enable the problem solver to perform its task.

There may be cases where there is a continuum of compliance and contradiction, rather than a binary decision (e.g., when predictions underlie some probability distribution). But, usually, there are natural thresholds that express tolerable deviations (from normal behavior, the design specification, etc.).

To be effective in the framework of consistency-based problem solving, completeness matters, i.e. its ability to detect all existing (or relevant) inconsistencies. Besides the fact that this can be expensive, model-based predictors can be inherently incomplete. A numerical simulation model (say, in Matlab) may appear as an appropriate solution in some cases (and even be readily available from engineering practice), but its fixed computational directionality may prevent the detection of all inconsistencies.

10.3.2 Validity of Behavior Modeling

The condition discussed above ensures that an inconsistency between the **model** and a description of some (real or hypothetical) situation is detectable. However, in order to draw safe conclusions about the **actual system**, the model has to represent its behavior in a valid way. While this seems pretty obvious, we can, and need to be, more specific. For consistency-based problem solving, it is essential that an inconsistency between a **model** and some criterion really indicates that the modeled **system** contradicts the criterion. In order to avoid spurious inconsistencies, we must postulate that **a model is guaranteed to be consistent with all situations the modeled real system can experience in reality**. As a consequence of this requirement, appropriate models tend to be conservative, for instance, by using the most generous tolerances of parameters. Of course, it can never be satisfied in an ideal way. The application context determines the scope of such really occurring situations, and, e.g., in circuit diagnosis, there is usually no need to include super conductivity at low temperatures in the model. However, the model must cover situations beyond the intended use of the component, e.g., a higher voltage caused by some defective transformer.

Again, this may appear obvious, but is sometimes hard to achieve and actually not fulfilled by many models in engineering, which are developed to work in a particular context and under certain environmental conditions.

10.3.3 Conceptual Modeling

Behavior prediction and consistency check refers to the **behavior** description, i.e. some mathematical, logical or other formalism that characterizes the state of the system. However, problem solvers refer to **concepts** of the real systems: components and their faults, design decisions, unwanted effects, unexpected substances and processes, etc. The solution space of models is spanned by these concepts, rather than by the mathematical, etc. expressions constraining the respective behavior, and the search and reasoning of the problem solver is performed in this space. Hence, these concepts and their relations have to be explicitly represented in model-based reasoning systems. Actually, this is lacking in most formalisms used in mathematical and engineering modeling, and this is where AI can make an essential contribution. This distinguishes, for instance, model-based diagnosis in AI from diagnosis systems in control engineering that perform a search in a space of mathematical models in order to find one that matches the observations (e.g., by means of parameter identification) without any representation of the physical structure of the device, its component faults, etc.

The decomposition of a real system into its entities (components, objects, relevant processes, ...) has to be made explicit and induces a structure of the behavior model. If this link between the relevant entities of the system and the behavior model is weak, then the conclusions that a model-based problem solver can draw at the conceptual level from a behavioral inconsistency are limited. If an equation solver only delivers the information that the entire system model is over-determined without any reference to a subset of component models that cause this, there is not much to be gained for localizing the fault.

It is clear that this feature is important for the efficient construction and maintenance of a model library.

10.3.4 (Automated) Model Composition

Having argued for the **decomposition** of a system model into fragments that correspond to the relevant constituents, we also need the opposite: the **composition** of model fragments in order to obtain a model of the overall system or subsystems. More precisely, what we need are algorithms for **automatically** composing system models. This is mandatory if the problem solver follows a generate-and-test strategy. If it generates a new hypothesis to be checked for consistency (say, a new combination of faults) then the generation of the respective model based on the model library must not involve the agent that usually composes models: a human modeler. Although the principle of modular and compositional modeling is not an invention of model-based reasoning, it is not straightforward and not supported in many modeling environments used in practice. For instance, although Matlab/Simulink provides means to organize a system model in a hierarchical manner as interlinked subsystem models, the lower level models cannot be arbitrarily combined because of the fixed computational directionality. Even if we model the same system, but start the computation from a different set of observed variables, the models of the subsystems are different and cannot be reused. In contrast, constraint systems ([13, 63] and Modelica [76]) are undirected and support compositionality.

10.3.5 Genericity

Compositionality of models is not only a matter of computational or structural aspects, such as directionality and compatible variable domains. The behavior models of the system constituents have to be stated in a context-independent manner in order to be usable in different contexts. Otherwise, the composed model will not cover the entire system behavior and violate validity as discussed in Section 10.3.2. For instance, if the scope of a task includes the occurrence of fault situations (as in diagnosis or FMEA), then a component model has to cover the response of the component to this faulty environment, which is one reason why many models developed for control purposes are not suited for model-based diagnosis. For instance, if a pipe is connected to a check valve, its model must nevertheless also cover a reversed flow in order to avoid wrong predictions and inconsistencies in case the check valve is broken. This principle has been termed “no function in structure” in [17]. For systems and variable-based models that treat some variables as exogenous, the requirement implies that the model must consider the entire Cartesian product of the respective variable domains. If it would not include the response of the component to some input, it would generate a spurious inconsistency if the respective situation appears.

Such sets of exogenous variables need not be unique for a single component. In a valve model, we can treat pressure at both sides as such a set and determine the flow from it. However, if the flow on one side is restricted to zero by a neighboring component (say, a clogged pipe), then it becomes an exogenous variable. Often, approaches to using causal models (e.g., causal graphs or Bayes’ nets) suffer from a similar deficiency, because the overall structure, the behavior of neighboring components, or certain assumptions are compiled into them. Also, the causal structure may change, even under normal behavior: the electric motor of a tram way can intentionally be turned into a generator and, hence, function as a brake. Even more frequently, faults modify the causal structure. Even if it is possible to capture all these variations in a causal graph, the model will hardly be compositional, and the effort of building complete causal models of large systems is prohibitive. Only ontologies that are based on local, independent causal interactions, such as qualitative process theory [34, 35] promise to provide a basis for model-based problem solving along these lines.

Limited genericity of the model fragments results in limited reusability and, as discussed earlier, reduces the application benefit.

10.3.6 Appropriate Granularity

Granularity refers to the degree of “resolution” of both the **structure** and the **behavior**. The structural granularity has to allow the reference to the concepts required by the task, e.g., the fault modes of the relevant components. For a compositional model, it is determined by the granularity of model fragments in the library, which may be more fine-grained than required. For instance, in on-board diagnosis, the set of observables is usually fixed, and all that matters for computation of diagnostics is the relation between these observables and the fault modes, while the model includes many unobservable internal and intermediate variables. In order to achieve a compact representation and efficient computation, as required by on-board diagnosis, it can make sense to transform the composite model appropriately [26].

The granularity of behavior descriptions, expressed, e.g., by the domains of variables, is determined by the purpose, namely to detect inconsistencies. For instance, if all values of a certain observable in one interval are consistent with the same set of models, it is not necessary to distinguish between them in the model. Because a fault can be characterized by causing a **significant deviation** from some intended behavior, tasks related to diagnosis and fault analysis even of continuous systems can often be based on qualitative models [81, 35, 4]. Since the required distinctions may depend on the task and the structure and intended function of the system and, hence, cannot be anticipated by the model fragments in the library, a composite model may have inappropriate domains. If they are too fine-grained, it may pay off to transform the composed model to a more abstract level [67].

There is a tension between the requirement of having compositional, generic, and reusable model fragments in a library and the necessity to achieve a compact representation of a model that is yet powerful enough for consistency checking and efficient computation. Violating one of them may eliminate the feasibility or at least the benefit of model-based systems in industrial applications. This is why research on multiple modeling and automated model transformation and compilation [69, 52, 21, 10] can make an important contribution.

10.4 Diagnosis

There is a huge variety of diagnostic tasks, and they may impose quite different requirements on modeling, model-based prediction and consistency check, the search algorithm, etc. The type of system and the practical context may emphasize different problems. On-board diagnosis on a vehicle has to be based on a fixed, and usually small, set of sensor values, while a fault in a power network generates an overwhelming burst of messages. Also, on-board diagnosis has to discriminate between different classes of faults according to their safety relevance and the resulting recovery actions, while diagnosis in the workshop only needs to identify the broken part in order to replace it. The latter case usually involves a number of testing activities, while a huge gas turbine in a plant does not allow interruptions for carrying out experiments. Most of the time, we are confronted with "post-mortem" diagnosis, but often, it is desirable to perform prognostic diagnosis in order to schedule maintenance before a failure occurs. And so forth.

Rather than outlining all specific answers to such specific requirements, we focus on the presentation of some principled and sufficiently general and influential approaches. We will identify the underlying assumptions that confine the scope of applicability. Even for some fundamental work, they were often left implicit, and sometimes, the authors even seem to be unaware of them.

We first describe consistency-based diagnosis with component-oriented models, whose idea has been the basis for the analysis in Section 10.2 and contains important principles and techniques, which partly carry over to other tasks. It represents probably the largest class of implemented systems and provides a systematic framework to the community for discussing variants and alternatives of the techniques.

Section 10.4.2 discusses the problem of performing diagnosis over time. We then outline an alternative concept, abductive diagnosis (Section 10.4.3) and consistency-based diagnosis using an alternative type of models, process-oriented diagnosis (Section 10.4.4).

10.4.1 Consistency-based Diagnosis with Component-oriented Models

The classical theory [62, 19] and realization of consistency-based diagnosis [20, 22, 24, 64] consider systems that contain a **fixed set of components**, *COMPS*, that interact in a **fixed structure**. Furthermore, it is assumed that a **disturbance** of the entire system is caused by a **malfunctioning** of one or more of these **components**. This includes the assumption that the entire system performs as intended if all components perform properly, i.e. the **well-designed system** assumption.

Diagnosis is then seen as the task to decide whether there are components that are not exhibiting their intended behavior (*fault detection*) and to determine which components work in a fault mode (*fault localization*) and in which fault mode they operate (*fault identification*).

Hence, each component C_i has a set of possible associated **behavior modes** $modes(C_i)$ (usually determined by the component type), and assigning one mode to each component provides an answer to a diagnosis problem.

Definition 10.1 (Mode assignment). *Let* $COMPS' \subseteq COMPS$.

$$\bigwedge_{C_i \in COMPS'} m_{j_i}(C_i), \quad \text{where } m_{j_i} \in modes(C_i)$$

is a mode assignment. It is called complete if $COMPS' = COMPS$.

$ok(C_i)$ always has to be an element of $modes(C_i)$ and characterizes uniquely the intended, normal behavior of the component. Modes are mutually exclusive,

$$m_{ji}(C_i) \wedge m_{ki}(C_i) \;\Rightarrow\; j = k$$

which also means that all modes different from $ok(C_i)$ represent some sort of faulty behavior:

$$\forall m_{ji}(C_i) \in modes(C_i) \setminus \{ok(C_i)\}, \quad m_{ji}(C_i) \Rightarrow \neg ok(C_i).$$

The model library *LIB* associates a behavior model with each mode:

$$m_{ji}(C_i) \Rightarrow model_{ji}(C_i).$$

If the models are stated in terms of variables, then *LIB* must also contain *domain axioms* for the variables, i.e., the (exclusive) disjunction of their possible values.

Then each mode assignment

$$MA = \bigwedge_{C_i \in COMPS} m_{j_i}(C_i)$$

together with the structural description *STRUCTURE*, which specifies the connections between components in terms of variables shared by the components, and the library *LIB* implies a behavior model *MODEL*(*MA*) of the entire system for the mode assignment *MA*:

$$LIB \cup STRUCTURE \cup \{MA\} \Rightarrow MODEL(MA)$$

Some papers use the term *system description* (*SD*) to refer to knowledge about the system without further specification. If we assume that there are no general restrictions on the possible mode assignments, we consider

$$SD = LIB \cup STRUCTURE$$

which has the disadvantage of obscuring the different nature of these elements: *LIB* usually contains domain-specific knowledge about the behavior of components, while *STRUCTURE* is system-specific.

In the following, we will always assume that modeling has led to a proper result, i.e., *SD* is consistent. If the modes of the components are assumed to be independent of each other, then also *MODEL*(*MA*) is consistent for every mode assignment *MA*. This requires valid models, as discussed in Section 10.3.2.

In this approach, model-based diagnosis is regarded as generation of system models that are consistent with the observations and amounts to generating hypotheses about the actually present behavior modes of the components. Therefore, we define

Definition 10.2 (Consistency-based diagnosis). *A complete mode assignment MA is a consistency-based diagnosis for a system description SD and a set of observations OBS if*

$$SD \cup \{MA\} \cup OBS \not\models \bot.$$

Fault detection

In particular, the assignment of *ok* to all components

$$MA_{OK} = \bigwedge_{C \in COMPS} ok(C)$$

specifies the normal behavior of the overall system:

$$LIB \cup STRUCTURE \cup \{MA_{OK}\} \Rightarrow MODEL_{OK}.$$

The well-designed-system assumption,

$$MODEL_{OK} \Rightarrow GOALS$$

turns the question whether the system behaves as intended into checking whether

$$SD \cup \{MA_{OK}\} \cup OBS \models \perp$$

which is realized by checking whether the resulting model is consistent with the observations:

$$MODEL_{OK} \cup OBS \models \perp.$$

Fault localization

If diagnosis is seen as fault localization, as, for instance, in [62, 20, 19], then this is related to another hidden assumption, namely that this information suffices to repair the broken system, which is true if **replacement of components** is the means for reestablishing the functionality of the system. (Sometimes, fault localization may be performed not in order to repair the system, but to identify flaws in manufacturing process.)

Fault localization is only interested in differentiating the broken components from the correctly working ones and, hence, aims at the special case of

$$modes(C) = \big\{ok(C), \neg ok(C)\big\}.$$

As stated above, if there are more specific fault modes, then they imply $\neg ok(C)$. A fault localization has to hypothesize the set of broken components consistently with the observations:

Definition 10.3 (Fault localization). *$FAULTY \subset COMPS$ is a fault localization for SD and OBS, if the mode assignment MA_{FAULTY}*

$$\bigwedge_{C \in FAULTY} \neg ok(C) \wedge \bigwedge_{C \in OK} ok(C)$$

with $OK = COMPS \setminus FAULTY$ is a diagnosis for SD and OBS. It is called minimal, if no proper subset of $FAULTY$ is also a fault localization.

This corresponds to the definitions in [20, 62, 19] (where fault localizations are called diagnoses and also *candidates*, because they might be refuted when additional observations are available) and is the basis for the *General Diagnosis Engine* (*GDE*) [20]. Minimal fault localizations are of practical interest because if a certain set of defective components suffices to explain the symptoms, why would we assume additional components also to be broken?

Computing (minimal) fault localization requires checking the consistency of the respective system models with the observations. If only the correct behavior is modeled, $\neg ok(C)$ has no model associated (which is equivalent to associating a model that does not impose any restrictions on the values of local variables). In this case, a search could be performed by eliminating the OK models of components from the entire model and checking the consistency of the remaining models. This approach which in practice might work in an exhaustive manner only for single or small sets of faults has actually been proposed in [12] as *constraint suspension*. However, there is a possibility for a more focused generation of fault localizations which has an intuitive basis: if the windshield wipers in our car do not work, we will focus our analysis on s small subset of components, such as their motor, the connecting cables, etc., but not consider, say, parts of the engine or of the braking system, because they do not influence the observed behavior of the car. Carried over to model-based diagnosis, this means that the observations may not simply be inconsistent with the complete system model, but with a model obtained from some partial mode assignment, which we call a *conflict*.

Definition 10.4 (Conflict). *Let $COMPS' \subset COMPS$ and*

$$MA = \bigwedge_{C_i \in COMPS'} m_{j_i}(C_i)$$

be a mode assignment such that

$$SD \cup \{MA\} \cup OBS \models \bot.$$

The negation of MA,

$$\bigvee_{C_i \in COMPS'} \neg m_{j_i}(C_i)$$

is called a conflict. *It is called minimal*, *if it is not implied by a different conflict*. *It is called* basic *if*

$$\forall C_i \in COMPS', \quad m_{j_i}(C_i) = ok(C_i) \vee m_{j_i}(C_i) = \neg ok(C_i)$$

and positive, *if*

$$\forall C_i \in COMPS', \quad m_{j_i}(C_i) = ok(C_i).$$

Since [19] considers only the two basic modes, a basic conflict corresponds to their definition of a conflict. Minimal conflicts are the most focused restrictions on the possible combinations of modes, and non-minimal conflicts do not provide additional information. Obviously, positive conflicts are important to fault localization, because they state that at least one of the mentioned components is broken. Even stronger, the following theorems [19] states that conflicts capture exactly the available information for fault localization, can replace $SD \cup OBS$ and be used to logically characterize the solutions.

Theorem 10.1. *Let MB-CONFLICTS be the set of all minimal basic conflicts for $SD \cup OBS$. FAULTY $\subset$ COMPS is a fault localization for $SD \cup OBS$ iff the respective mode assignment is consistent with the minimal basic conflicts*:

$$\textit{MB-CONFLICTS} \cup \{MA_{FAULTY}\} \not\models \bot.$$

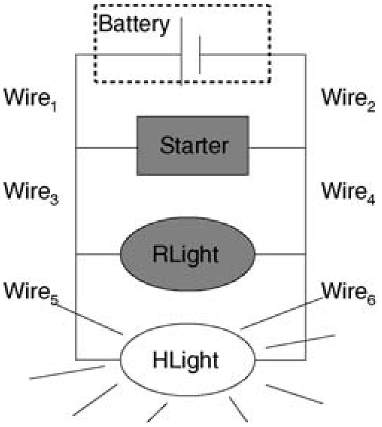


Figure 10.3: A simple diagnostic problem: the head lights are lit, while the rear lights are not, and the starter does not work.

Theorem 10.2. *FAULTY $\subset$ COMPS is a minimal fault localization iff the mode assignment*

$$\bigwedge_{C \in \mathit{FAULTY}} \neg ok(C)$$

is a prime implicant of the positive minimal conflicts.

A prime implicant of a set of formulas is a minimal conjunctive clause of literals (in our case representing $ok(C), \neg ok(C)$) that entails each formula in the set. This captures the intuition that (minimal) fault localizations have to satisfy exactly each (minimal) disjunction of suspect components. One way to obtain them is to compute minimal hitting sets of the components contained in the minimal positive conflicts [62, 38]. A hitting set of a set of sets $\{A_i\}$ is defined by having a nonempty intersection with each A_i. As an illustrative example, we reconsider a slightly modified example from [64]: the starter of a car and its rear lights and front lights supplied by a battery in parallel (Fig. 10.3). However, we observe that the rear lights are dark and the starter does not work, while the head lights are lit. We assume that the library contains (only) models of the correct behavior of the involved components: a battery supplies voltage, wires act as electrical connectors, and, if supplied with a voltage drop, light bulbs are lit and the starter acts. Such models for the battery, the starter and Wire_1, Wire_2 will predict all together that the starter is active, contradictory to the observation:

$$ok(\text{Battery}) \wedge ok(\text{Wire}_1) \wedge ok(\text{Wire}_2) \wedge ok(\text{Starter}) \Rightarrow active(\text{Starter}),$$

$$OBS \Rightarrow \neg active(\text{Starter}).$$

This yields a conflict

$$\text{Conflict}_1 \equiv \neg ok(\text{Battery}) \vee \neg ok(\text{Wire}_1) \vee \neg ok(\text{Wire}_2) \vee \neg ok(\text{Starter})$$

which is positive and minimal. Similarly we obtain

$$\begin{aligned}\text{Conflict}_2 \equiv\ & \neg ok(\text{Battery}) \vee \neg ok(\text{Wire}_1) \\ & \vee \neg ok(\text{Wire}_2) \vee \neg ok(\text{Wire}_3) \vee \neg ok(\text{Wire}_4) \vee \neg ok(\text{RLight}).\end{aligned}$$

Furthermore, the lit head lights imply the existence of a voltage drop which should also cause the rear lights to be lit, leading to

$$\text{Conflict}_3 \equiv \neg ok(\text{HLight}) \vee \neg ok(\text{Wire}_5) \vee \neg ok(\text{Wire}_6) \vee \neg ok(\text{RLight}).$$

Analogously, we find

$$\begin{aligned}\text{Conflict}_4 \equiv {} & \neg ok(\text{HLight}) \vee \neg ok(\text{Wire}_5) \vee \neg ok(\text{Wire}_6) \\ & \vee \neg ok(\text{Wire}_3) \vee \neg ok(\text{Wire}_4) \vee \neg ok(\text{Starter}).\end{aligned}$$

In fact, these are all minimal and positive conflicts. As a side-remark, the last two conflicts are only derived if the predictor is complete enough to reason not only in the causal direction, but also draw conclusions from the effect, namely the lit head lights. The mode assignment

$$\neg ok(\text{RLight}) \wedge \neg ok(\text{Starter})$$

implies all conflicts and is minimal, hence a prime implicant of all positive minimal conflicts. Thus,

$$\{\text{RLight}, \text{Starter}\}$$

is a fault localization, in accordance with our expectation.

At this point, we emphasize, that the described approach allows for

- fault localization with models of correct component behavior only, i.e. without any restriction on the possible faulty behaviors,
- localizing multiple faults.

This is an advantage over systems based on empirical symptom-fault associations, which require explicitly known faults and face natural limitations on known symptoms of multiple faults.

If the library does not contain fault models, there is no way to refute $\neg ok(C_i)$, all basic conflicts are positive ones, and extending a fault localization by additional components also yields a fault localization. In general, we have [19]:

Theorem 10.3. *For each fault localization FAULTY* $\in$ *COMPS every superset FAULTY*$'$ $\supset$ *FAULTY is also a fault localization iff all basic conflicts of SD* $\cup$ *OBS are positive.*

In this case, the minimal fault localizations are a compact representation of all fault localizations, namely as a lower bound in the subset lattice of *COMPS*.

Fault localization with fault models

When taking a second look at the example, we notice that, while we are satisfied with the fault localization {RLight, Starter}, we would not consider, for instance, its superset {RLight, Starter, Battery} as a reasonable fault localization, despite Theorem 10.3. Furthermore, we notice that there are many more prime implicants of the four conflicts, namely 21, and among them are, for instance,

$$\neg ok(\text{Wire}_1) \wedge \neg ok(\text{Wire}_5)$$

and

$$\neg ok(\text{Battery}) \wedge \neg ok(\text{HLight})$$

which we may not want to accept as a plausible fault localization! The reason why we find them implausible lies in the fact that the observations contradict the expected faulty behavior of the suspected components: the head lights would not be lit if they were broken. While **not requiring** models of faulty behavior, fault localization may become more focused and realistic when exploiting fault models.

One way to do this has been introduced in GDE$^+$ [64] by associating models with fault modes and *physical negation* axioms

$$\neg ok(C_i) \ \Rightarrow \ \bigvee_j fault_{ji}(C_i)$$

in order to express that the negation of the ok behavior in physical systems does not lead to totally unrestricted behavior, but to a certain set of unintended behaviors that can still be described. If the fault modes of some component C_i can be refuted by the observations in conjunction with a mode assignment to other components, MA, or directly, i.e. ($MA = \emptyset$), i.e. for all i

$$SD \cup \{MA \wedge fault_{ji}(C_i)\} \cup OBS \models \bot$$

then C can be exonerated in this context:

$$SD \cup \{MA\} \cup OBS \models ok(C_i)$$

by means of the physical negation axiom. By adding meaningful fault models for the components in our example (expressing "broken lights are never lit", etc.) and the physical negation rule, the only remaining fault localization will the plausible one. However, if some exotic faults are ignored in our model, the proper fault localization may be missed. For instance, if Wire$_1$ were open, while Wire$_5$ is open, but shorted to source at the end towards the head lights, the fault localization {Wire$_1$, Wire$_5$} would make sense. We may try to account for such unforeseen faults by introducing a fault mode with unspecified behavior. But this could not be refuted and the exoneration not be concluded, which means that fault localization would also be not affected by the use of the other fault models. We need some additional concepts which will be discussed in the following subsection.

As an alternative, Friedrich et al. [31] propose to represent situations that are physically impossible (under all modes) instead of the various faults (e.g., that head lights without voltage are never lit).

With the introduction of fault models and, hence, the possibility of conflicts that are not positive, the minimal fault localizations are no longer the generators of all fault localizations. Intuitively, this is because a minimal fault localization may become inconsistent if a fault mode of another component is added. In our example, the fault localization {RLight, Starter, HLight} is a superset of {RLight, Starter}, but inconsistent (because a fault in HLight directly contradicts the observations).

To obtain a generating set for the case of fault models, the concept of *kernel diagnosis* was introduced [19].

Definition 10.5 (Kernel diagnosis). *A kernel diagnosis is a minimal partial mode assignment MA_k with the property that every mode assignment that extends it is consistent with $SD \cup OBS$, i.e.*

for all consistent MA holds
if MA entails MA_k
then MA is a diagnosis of $SD \cup OBS$.

In other words, the modes of the components not mentioned in MA_k do not matter. Obviously, all fault localizations are obtained from an extension of some kernel diagnoses. Also, the kernel diagnoses can be characterized as prime implicants of all minimal conflicts.

Fault identification

Besides helping to refine fault localization, fault models provide the basis for identifying which particular component faults may be responsible for the disturbed system behavior. If the list of behavior modes contains specific faults of a component (type), then the diagnoses according to the definition given above are the answer to the task of fault identification.

However, the inclusion of explicit fault models in *SD* is a qualitative jump from a single system model (of the ok behavior) to a large space of models, corresponding to all possible mode assignments. This is important from both a technical and an application point of view.

Technically, it implies that many system models may have to be checked for consistency with observations, and for conflict-driven approaches, it means that the space of minimal conflicts grows. Fortunately, the application perspective implies that most of the mode assignments are not interesting and many conflicts need not be discovered. To most diagnosis applications, it is not interesting to **characterize** the space of **all** diagnoses, but to **compute** the most **relevant** ones. This is because its purpose is to provide information just enough to restore the functionality. Therefore, of course, what makes a diagnosis relevant depends on the type of system and its application context. But to be of practical interest, diagnostic theories and systems should provide generic means to express some ranking of the expected diagnoses and algorithms to effectively and efficiently compute the best ones under such a ranking. Unfortunately, there have not been as many theoretical contributions to this important area as to the logical characterization and approaches assuming exhaustive computation.

The applied principle of Occam's razor, namely not to assume more components to be broken than necessary, is usually a fundamental criterion we would like to preserve for fault identification, as well.

Definition 10.6 (Minimal diagnosis). *A diagnosis MA for $SD \cup OBS$ is a minimal diagnosis, iff the corresponding fault localization*

$$FAULTY := \{C_i \in COMPS \mid ok(C_i) \notin MA\}$$

is minimal.

This set of minimal diagnoses may still be large and ignore additional ranking criteria. Both a broken (open) light bulb and its pin being shorted to ground may explain why the bulb is not lit, but the shorted fault may be much more unlikely and, hence, only considered if the other one has been ruled out. We can define such a general preference on the modes of a component.

Definition 10.7 (Preference on modes and mode assignments). *A mode preference for C_i is a partial order "$\geqslant$" on $modes(C_i)$*:

$$\geqslant \subseteq modes(C_i) \times modes(C_i),$$

where $ok(C_i)$ is the maximal element and an unknown fault mode $unknown(C_i)$, if present, is the minimal element:

$$\forall m_j(C_i) \in modes(C_i) \setminus \{ok(C_i)\}\colon ok(C_i) > m_j(C_i),$$
$$\forall m_j(C_i) \in modes(C_i) \setminus \{unknown(C_i)\}\colon m_j(C_i) > unknown(C_i).$$

"$>$" is defined as

$$x > y \quad :\Leftrightarrow \quad x \geqslant y \wedge \neg(y \geqslant x).$$

This induces a preference on mode assignments: *for*

$$MA = \{m_{j_i}(C_i)\},$$
$$MA' = \{m'_{j_i}(C_i)\},$$

we define

$$MA \geqslant MA' \quad :\Leftrightarrow \quad \forall i\ m_{j_i}(C_i) \geqslant m'_{j_i}(C_i).$$

Definition 10.8 (Preferred diagnosis). *A diagnosis MA is a preferred diagnosis*, *if there is no diagnosis MA' that is strictly preferred over MA*

$$\forall MA'\ MA' \geqslant MA \quad \Rightarrow \quad MA' = MA.$$

Intuitively, the definition expresses, that a certain fault mode $m_j(C_i)$ should appear in a preferred diagnosis MA if

1. all mode assignments that are obtained by MA replacing $m_j(C_i)$ in MA by a strictly preferred mode $m'_j(C_i) > m_j(C_i)$ are not a diagnosis, and, of course,
2. MA is a diagnosis.

In order to characterize preferred diagnoses, [24] uses default logic [61, 5]. A (normal) default is an inference rule of the form

$$a : b/b$$

which expresses, intuitively, "If a is true, and it is consistent to assume b is true, then b holds". A default theory is a pair (D, P), where P is a set of classical formulas and D a set of defaults. Since defaults may exclude each other mutually, there are different (maximal) sets of defaults applicable, which leads to different sets of conclusions, called *extensions*.

For instance, assuming a certain mode of a component, we cannot associate another mode of the same component that might also be consistent. Indeed, we can encode the rule that $m_j(C_i)$ should be assumed only if all its strictly preferred predecessors

$$pre_j(C_i) := \{m_k(C_i) \mid m_k(C_i) > m_j(C_i)\}$$

have been refuted, and if $m_j(C_i)$ can be consistently assumed as a default

$$def_{ij} \equiv \bigwedge_{m_k(C_i) \in pre_j(C_i)} \neg m_k(C_i) : m_j(C_i)/m_j(C_i).$$

Especially, the ok behavior will be assumed first:

$$: ok(C_i)/ok(C_i)$$

The following theorem [24] captures the intuition that these preference defaults determine the preferred diagnoses:

Theorem 10.4. *Let DEF* $= \{def_{ij}\}$ *be the set of all preference defaults. MA is a preferred diagnosis if*

$$Cn(SD \cup OBS \cup MA)$$

is an extension of the default theory $(DEF, SD \cup OBS)$. *Here*, $Cn(.)$ *denotes the deductive hull*:

$$Cn(P) := \{p \mid P \vDash p\}.$$

The theorem provides the logical characterization of (preferred) diagnoses for fault identification and contains as a special case, namely $modes(C_i) = \{ok(C_i), \neg ok(C_i)\}$, the characterization for fault localizations given in [62]. The theory was implemented as the *Default-based Diagnosis Engine (DDE)* [25] which generates the successor mode assignments for the refuted diagnosis hypotheses according to the preference relation and checks their consistency only if all strictly preferred diagnoses have been refuted. This means, in particular, if an unknown fault is included, it will only be considered if no other fault mode survives the consistency check, but its existence prevents exoneration as performed in GDE^+.

DDE's preferences are local to each component and only an ordering. It does not use preferences among modes of different components and, hence, does, for instance, not order single faults involving different components. A refinement can be obtained by exploiting a global scale for ranking of modes, such as failure probabilities. In *SHERLOCK* [22], mode assignments are checked for consistency in the order of their probability which is obtained from the probabilities of modes (assuming their independence). Starting with a-priori probabilities, *SHERLOCK* recomputes probabilities when new conflicts have been detected. Unknown faults can be included, usually with low probability, and termination criteria specified, e.g., as a function of the probabilities of the diagnoses obtained so far. Although there is no formal characterization, it should be clear that *SHERLOCK* generates a subset of the preferred diagnoses if the preference is the order induced by mode probabilities. The core of this technique is fairly general and has been introduced as *conflict-directed* $A*$ search [86].

10.4.2 Computation of Diagnoses

Since diagnosis is formalized as finding a model that is consistent with the observations, one might (and some authors do) suggest using any (efficient) generic algorithm that generates a solution for

$$SD \cup \{MA\} \cup OBS$$

such as constraint satisfaction algorithms [13, 63] and SAT-solvers. However, while many such algorithms produce **some single** solution quite efficiently, their naive use may ignore requirements and context of the real task. The same holds for the, usually infeasible, attempt to compute the set of **all** diagnoses. Diagnosis in the real world is not interested in a single arbitrary solution, but in **finding a set of diagnoses that fulfill some criteria dictated by the practical context of the task**. Such criteria vary and can be quite complex. Minimality (with respect to cardinality or set inclusion) of diagnoses is only one example, which is independent of domain and task. In reality, the relevance criteria for diagnoses are mainly determined by the ultimate objective, namely re-establishing the proper system behavior at minimal cost and risk, and, hence, may vary with the means and restrictions for reaching the objective (see the discussion in Section 10.6). Focusing on the most likely or "preferred" faults as done in *SHERLOCK* [22] and GDE^+ [24], respectively, reduces the risk of fixing the wrong component and, thus, the average cost. In some applications, certain highly critical faults may have to be explicitly ruled out (e.g., to select appropriate recovery actions based on on-board diagnosis of vehicles).

Another important requirement in many applications is due to the fact that computing diagnoses from observations is not a one-shot activity, but happens multiple times in a process of gathering information through testing and observation (see Section 10.5). This has to be reflected by the requirement for algorithms that support an efficient **incremental** computation of diagnoses when the set of observations is extended.

The design of a diagnosis algorithm has to reflect a number of choices imposed by the respective application:

- The task: fault **localization** vs. fault **identification**.
- The models: existence or non-existence of **fault models**.
- Fault models: existence or non-existence of an **unknown fault** (with unrestricted behavior).
- The result: criteria for the **relevance of diagnoses** to be produced (rarely all).

In the theories presented above, the concept of conflicts played an important role in characterizing the solution space. We discuss some aspects of exploiting conflicts in some more detail.

Computing fault localizations/diagnoses from conflicts

Theorem 10.2 suggests a two-step computation: first compute all minimal (positive) conflicts, then compute their prime implicants to obtain fault localizations. This is feasible and useful, if only the *OK* behavior is modeled. *GDE* [20] is the archetype of this solution. The presence of fault models modifies the set of minimal positive conflicts, if the *physical negation* rule is applied (i.e. the set of fault models is considered complete and does not contain an unknown fault as in GDE^{+} [64]). For instance, in our example:

$$\mathit{Conflict}_3 \equiv \neg ok(\mathit{HLight}) \vee \neg ok(\mathit{Wire}_5) \vee \neg ok(\mathit{Wire}_6) \vee \neg ok(\mathit{RLight})$$

is reduced to

$$\mathit{Conflict}_3 \equiv \neg ok(\mathit{Wire}_5) \vee \neg ok(\mathit{Wire}_6) \vee \neg ok(\mathit{RLight})$$

by the non-positive conflict

$$\neg \mathit{broken}(\mathit{HLight})$$

(which is obtained from the observation that HLight is lit) and the physical negation rule:

$$\neg ok(\mathit{HLight}) \Rightarrow \mathit{broken}(\mathit{HLight}).$$

The introduction of fault models implies the step from a single system model (the *OK* model) to a large set of models (for all mode assignments). This is a qualitative leap, which usually makes a complete check of all mode assignments infeasible.

Computing kernel diagnoses

The concept of kernel diagnoses, introduced in Section 10.4.1, is attractive from a theoretical point of view, because it provides a generator for the set of all diagnoses in case of the existence of fault models. However, it does not offer the basis for any practical solution, because it requires an unrestricted consistency check of mode assignments. Also, many of the kernel diagnoses may be completely irrelevant to any practical consideration. We illustrate this by the following example. Consider, say, 17 "Equal components" Equal_i in series which have the modes

$$\begin{aligned} ok(\text{Equal}_i) &: in_i = out_i, \\ neg(\text{Equal}_i) &: in_i - 1 = out_i, \\ pos(\text{Equal}_i) &: in_i + 1 = out_i \end{aligned}$$

and the observations

$$\begin{aligned} in_1 &= 0, \\ out_{17} &= 1. \end{aligned}$$

Then there exist 17 singleton fault localizations, namely $\{pos(\text{Equal}_i)\}$, which are the interesting ones to focus on under practical considerations. The space of all fault localizations is given by all subsets of *COMPS* with odd cardinality. As a consequence, the set of kernel diagnoses is identical to the set of all fault localizations, which means, in particular, all of them are complete mode assignments. From a computational point of view, the example also illustrates that the set of non-positive conflicts is large namely the set of all subsets of *COMPS* with even cardinality and the empty set, and that determining them requires checking all complete mode assignments (but then, you have the fault localizations directly).

In summary, an exhaustive computation of conflicts rarely lends itself to a computational solution (except for fault localization with *OK* models only). However, there is no interest in computing all diagnoses, anyway.

Search for diagnoses and the exploitation of conflicts

The response to this insight is to organize the generation of relevant diagnoses as search, instantiating and checking mode assignments only after checking those with higher relevance. Given a criterion for (potentially dynamically) ordering mode assignments according to their importance, one could perform some best-first search in the space of mode assignments in a hypothesize-and-test cycle in a straightforward manner. However, (minimal) conflicts provide a powerful means to improve the efficiency of the search. This is due to the fact that a model of a mode assignment that does not satisfy all existing (minimal) conflicts does not need to be instantiated and checked for consistency with the observations. Stated differently, after each detection of a new (minimal) conflict, the search space can be pruned by eliminating all mode assignments that imply the respective inconsistent partial mode assignment (i.e. the negation of this conflict).

In *GDE*$^+$ [24], the preference defaults serve two purposes: on the one hand, they encode the ordering of the modes and ensure that a mode of a component is only considered for consistency checking in a context if all more preferred modes have been refuted. On the other hand, it will not be checked, if it is already known to be inconsistent because it is subsumed by some previously detected inconsistency. *SHERLOCK* [22], which checks mode assignments according to their probabilities, also exploits conflicts to prune the space of mode assignments. This principle has been generalized to *Conflict-directed A** [86] for cost functions satisfying certain criteria.

Determining (minimal) conflicts

Exploiting conflicts for computing fault localizations and pruning the search space requires that the consistency check delivers more than a Yes/No answer for a complete mode assignment. It has to identify partial mode assignments that generate the inconsistency, and the smaller they are, the stronger is the impact on the accuracy of the computed fault localization and on search space pruning. The "classical" way of finding conflicts (as in *GDE*, *GDE*$^+$, and *SHERLOCK*) is by means of a **propagation-based predictor** interfaced to some dependency-recording mechanism (e.g., exploiting an *Assumption-based Truth-Maintenance System, ATMS* [14]). Whenever a contradiction (two conflicting values of a variable) is detected, the underlying behavior modes that derived it together can be determined. Incompleteness of the predictor may lead to missing (minimal) conflicts and, hence, suboptimal fault localization (although the proper one will never be falsely refuted). However, while this works for some systems, such as combinatorial circuits, there is a vast space of system models for which propagation is highly incomplete or does not derive anything (resistive electrical circuits, hydraulic systems, ...). In this case, other more complete algorithms for consistency checking are needed, and if generic efficient ones are used (CSP, SAT, ...), then their utility depends on whether and to what extent they can deliver (minimal) conflicts.

Pre-compilation of models

If one does not use dependency recording or some equivalent technique, the alternative is to check partial mode assignments for consistency in order to find conflicts. But this is a large space, and one would want to anticipate which assignments can possibly lead to the detection of a conflict. This means to decompose the system into chunks in a way that checking these respective partial mode assignments can possibly lead to a conflict. The analysis needed for such an approach, which may be called **conflict-oriented model decomposition** [56], has to reflect the structure of the system **and** the set of observable variables. Intuitively, the task is to find sets of observations that partition the system model into subsystems that can become over-determined, which often requires to make certain assumptions about the model (e.g., linear functions). There are a number of caveats. Firstly, the approach is obviously only suited for applications where the set of possible observables is fixed (and not too large), an assumption that can be valid for online-diagnosis of monitored or controlled systems. Secondly, the potential conflicts can comprise quite different subsets of components for different mode assignments, and even for different states and inputs of the system. Performing the analysis exhaustively for all cases, particularly under the presence of fault models seems prohibitive. Hence, thirdly, if we use purely structurally oriented algorithms, we may fail to find the minimal potential conflicts.

There are other proposals to compile system descriptions in order to achieve better performance at diagnosis runtime. Ultimately, only the interdependencies between observable variables and the mode assignments matter, whereas the overall system model may contain many more intermediate and unobservable variables, especially due to the fact that the model is a compositional one. A straightforward step is, therefore, to eliminate all unobservable variables from the model. This works best if the set of observable variables is fixed (and small), as, for instance, in on-board diagnosis and monitoring systems, where the set of observables is determined by the existing sensors [26]. This has enabled the generation of a model-based on-board diagnostic system, that runs on an actual control unit of a passenger vehicle [74]. Darwiche [10] proposes to compile a system description into a special form (negation normal form) in order to achieve better performance for diagnosis tasks.

Obviously, for all such solutions holds that the complexity of the task is shifted into the compilation step which even may become intractable.

Hierarchical models

Another option is to represent the system to be diagnosed by a hierarchical model and apply the described techniques at each level to those subsystems that have been determined as suspects at the higher level. This keeps the number of components and, hence, the size of mode assignments and conflicts small. (See, e.g., [48].) While a solution along these lines is theoretically straightforward, in practice it comes at considerable cost and raises some problems. Obviously, we need models of subsystems above the level of elementary components. There are two ways to obtain them: automatically or "by hand". The latter option, though feasible in some cases, increases the modeling effort. The bad part is that only the models of the very bottom layers can be expected to be reusable, the rest is likely to be system-specific. Therefore, in most applications, the effort of creating models of higher-level components (which are hardly re-usable) manually will probably kill the economic benefit of a model-based solution. An automated solution is needed.

The reductionist approach implies that we can obtain the behavior models of the subsystems in a bottom-up fashion as the composition of the models of its components, which means we face the task of automated model compilation (e.g., by transforming a constraint network to a single constraint relating state and interface variables of the aggregate and covering all observable variables). If we would like to exploit fault models not only at the lowest level to improve fault localization, we have a complexity problem, because we have to generate not only the ok model of the aggregate, but also its fault models, which would mean compiling models of all or a selected set of mode assignments. An option is to focus on single faults (or the most probable ones) and capture the rest by an unknown fault mode of the aggregate. Still the result can be many fault modes for the aggregate. Often, they can be conveniently summarized by a smaller set of fault modes in a more abstract representation, but generating such abstractions automatically is a serious challenge to automated modeling—or we are back at manual modeling.

10.4.3 Solution Scope and Limitations of Component-Oriented Diagnosis

Although what has been surveyed so far in this section has often been considered as theories and solutions to **the** task of diagnosis based on first principles, it turns out to be a very specific one. We need to be aware that there are a number of underlying assumptions and limitations that confine the scope of applicability from a practical perspective.

- **Fixed, well-specified set of components**: many systems in process industries (e.g., in chemical plants) and, even more so, natural systems cannot be modeled conveniently as a set of components.

- **Known, fixed structure**: For the types of systems just mentioned, this is also not satisfied. And in some devices, we might have to consider the processed objects as components, such as sheets in a copier.

- **Well-designed system**: This assumes the intended functionality ("*GOALS*") is implied by the system with correct components. This is even not given for many carefully designed artifacts: often, the parameter tolerances of components in a circuit may well allow an unintended behavior, which is just not happening due to the statistical distribution of parameter deviations. And ecological systems are not designed anyway and, hence, always require an explicit representation and consideration of *GOALS* [41].

- **Component faults only**: Disturbances of the system behavior are always caused by a fault of one of the known components. However, often, the cause of a malfunction is due to some additional, unexpected object, substance, or agent intruding the system.

- **No structural faults**: Even if the structure of the correct device is well-specified, the fault may lie in a violation of this structure (e.g., a bridge fault) [16].

- **"Crisp" faults**: although the models of different faults may overlap, there is the assumption that the presence of a fault is a yes/no decision. In order to incorporate degradation and "soft" faults, one would have to introduce thresholds that, perhaps artificially, distinguish a tolerable degradation from a real fault.

But there are some **non-assumptions**, contrary to what is sometimes believed:

- Sometimes, it is believed that consistency-based diagnosis can only work with **specific modeling formalisms**, e.g., models that are expressed in, or can be transformed into, logical formulas, such as finite constraints. Engineering models do not come as logical formulas. However, the principles underlying consistency-based diagnosis are general. As discussed in Section 10.3, any modeling formalism that is suited to capture the diagnosis-relevant behavior aspects of component modes and that has some notion of and mechanisms for checking the consistency of a model with observations can be used. This includes numerical models and simulators, provided there is a way to avoid the creation of spurious inconsistencies due to noise, model inaccuracy, measurement imprecision, etc. Also, if computation is fixed to one direction (from "input" to "output"), they have to reflect the available observations what makes the system models specific and reduces their reusability, and they may suffer from incompleteness regarding the detection of all conflicts (because this may require inferences starting from outputs).

- In particular, it is often assumed that only **static system** behavior can be diagnosed. This is not true, since neither the theory nor the technical principles prevent the use of models that describe the dynamic behavior and of observations that capture the system evolution over time. Still, the temporal dimension introduces some additional problems and specific answers, which will be the subject of the next section.

Furthermore, it should be pointed out that there is a useful generalization of the theory and the techniques if we replace "behavior modes" by "states", where a state is the assignment of a value to a state variable of a component (in analogy to assigning a particular mode to a component). This way, hypotheses not only about the occurrence of faults, but also about the internal states of a system can be generated [84]. However, there is no general preference criterion (like minimality for sets of faulty components) for states, although, perhaps, for state changes.

10.4.4 Diagnosis across Time

If observations are available not just for one snapshot of system behavior, but for a whole observation period, this may strengthen the basis for diagnosis, but also triggers some special problems to solve. Extending the basic definitions appropriately is not too difficult. First, we have a **history** or sequence of observations

$$OBSH = \{OBS_i\} = \{\{obs_{ij}(t_i)\}\}$$

related to a finite set of time points t_i in some time interval of interest, $t_i \in I_\omega$. Secondly, not only the behavior of the system to be diagnosed may evolve over time, but also the behavior **modes** of components may change over time, i.e. faults may occur and also disappear. Therefore, in the general case, a diagnostic hypothesis is no longer **one** mode assignment, but a **history** of mode assignments

$$MH = \{(MA_k, I_k)\}, \quad \bigcup_k I_k = I_\omega, \quad MA_k \neq MA_{k+1},$$

where MA_k is a mode assignment that holds for all time points in some interval $I_k = (t_k, t_{k+1}) \subset I_\omega$, that is consistent with the observation history (see, e.g., [33]).

Definition 10.9 (Consistency-based temporal diagnosis). *A history of complete mode assignments*

$$MH = \{(MA_k, I_k)\}$$

in I_ω is a consistency-based temporal diagnosis for a system description SD and an observation history

$$OBSH = \{\{obs_{ij}(t_i)\}\}$$

in I_ω if

$$SD \cup MH \cup OBSH \nvDash \bot.$$

This concept of a temporal diagnosis subsumes the static version in the sense that for each observation point, the mode assignment must be a diagnosis according to Definition 10.2.

Definition 10.10 (State-based diagnosis). *A mode history*

$$MH = \{(MA_k, I_k)\}$$

is a state-based diagnosis for

$$OBSH = \{\{obs_{ij}(t_i)\}\}$$

if
for all t_i holds
if $t_i \in I_k$
then MA_k is a diagnosis for SD and OBS_i.

Lemma 10.1. *If $MH = \{(MA_k, I_k)\}$ is a temporal diagnosis of SD and*
$OBSH = \{\{obs_{ij}(t_i)\}\}$,
then
MH is a state-based diagnosis for SD and OBSH.

In other words, being a state-based diagnosis is a **necessary** condition for obtaining a temporal diagnosis. Amazingly enough, it is also a sufficient condition for an interesting and large class of systems and tasks, as discussed in the following.

Whether this holds strictly, depends also on the available observations, because some *sequence-constraints*, i.e. restrictions on the possible transitions between states, may compensate for limited observability. Let us illustrate this by a trivial example. Assume we parked our car in a street in San Francisco with considerable and applied the park brake. When we return 10 minutes later, we find the car is no longer at the place where we left it, but 50 m down the street in front of a wall (with some dents). We certainly suspect that the park brake did not do its job, despite the fact that the car was at stand-still when we left, but also the car in front of the wall is perfectly consistent with a well-functioning park brake. However, we can conclude that, since the positions are different, there must have been an unobserved state in between, where the speed of the car was non-zero which contradicts the *OK* mode of the park brake. This case shows that the *sequence-constraints* can compensate for gaps in observations in two ways: gaps in time (by conclusions for unobserved states) and regarding observable variables (esp. derivatives, here: the speed).

Computation of temporal diagnoses

The example does not only illustrate that the *sequence-constraints* can be essential to diagnosis, it also sheds a light on the implication for computational considerations; We did not have to simulate the vehicle's behavior under *OK* mode and the broken-park-brake mode to obtain a conclusion.

Instead, we inferred the existence of a state with $speed > 0$, which directly contradicts the *OK* model, while being consistent with the fault model. This illustrates: even if we cannot drop the temporal aspects from the consistency check of different mode assignments MA_j

$$SD \cup \{MA_j\} \cup OBSH,$$

which means

$$\mathit{state\text{-}constraints}_{ji} \cup \mathit{sequence\text{-}constraints} \cup OBSH,$$

without loss, this still does not force us to simulate the model of every candidate mode assignment MA_j, which is likely to be impossible anyway (e.g., in our example, we do not know when the car started to move and how). Instead, we can apply *sequence-constraints* first, to complement the observed history by indirect observations

$$OBSH \cup \mathit{sequence\text{-}constraints} \models OBSH_{ext}$$

and then perform consistency checks of

$$\mathit{state\text{-}constraints}_{ji} \cup OBSH_{ext}.$$

The computational advantage of the second solution is tremendous, since we avoid simulation of many fault hypotheses, apply *sequence-constraints* only once and perform cheaper consistency checks on states only. The most common application and exploitation of this approach is the computation of derivatives from observations to avoid simulation and obtain equivalent results (as analyzed and confirmed in [6] for numerical models). In essence, the good message is: **diagnosis of dynamic systems does not require simulation.**

If we perform state-based diagnosis of persistent faults as described above, then the mode assignment in the temporal diagnosis has to be a prime implicant of the union of all sets of conflicts detected at the various time points (or, rather, the minimal elements of this union). Hence, diagnoses can be computed incrementally by adding newly detected (minimal) conflicts for each observed snapshot. For systems that perform a best-first search, such as *SHERLOCK* and *DDE*, the set of diagnoses obtained from one snapshot (e.g., the most probable or the preferred diagnoses) forms the set of mode assignments to be checked against the observations for the next snapshot. Whenever, based on these checks, some diagnoses are refuted and new ones are generated, these also ought to be checked against all previous snapshots.

State-based vs. simulation-based diagnosis

When confronted with the necessity to diagnose a system whose behavior is observed and changes over time, the immediate consequence seems to be that one has to simulate the behavior under different mode assignments and check for consistency with the actual tracked behavior ([15] is an examples of such a solution). Triggered by the observation that in applications of consistency-based diagnosis conflicts always were generated from observations stemming from one snapshot [23], the analysis revealed that the underlying reasons are quite fundamental and lead to a fairly general characterization of preconditions for being able to refrain from simulation without affecting the quality of the resulting diagnoses [71].

The key consideration is that many computational modeling formalisms decompose a description of the temporal evolution of a system into a set of restrictions that hold for the state at each time point and a part that restricts the set of possible sequences of such states:

$$\mathit{model} = \mathit{state\text{-}constraints} \cup \mathit{sequence\text{-}constraints}$$

For instance, in a numerical simulation system, the former one is an ordinary differential equation, while the latter is incorporated in the integration algorithm. In this case, the specificity of *model* of a particular mode (assignment) is captured by the first part only, while the second part represents general laws that apply to all models, namely the laws of continuity, derivatives, and integration, and is shared by all possible behaviors an their models. As a consequence, any observed behavior of a particular mode assignment will be consistent with the model, if and only if it is consistent with its *state-constraints*. This provides an intuition for why checking the individual observation snapshots for consistency with the *state-constraints* suffices for diagnosis, and applying *sequence-constraints* and simulation can be avoided. However, if the observations have gaps, i.e., miss a state of the actual behavior, or the set of observable variables is too small to reveal an inconsistency, then exploiting *sequence-constraints* in simulation might compensate for it, because they could infer information about an intermediate state or additional information about a partially observed state (e.g., about derivates). This consideration can be turned into a rigorous argument [71] and a foundation for solutions of high importance to industrial applications, especially if faults are persistent. Faults are persistent if they do not vanish without repair (such as leakages or broken bulbs).

Definition 10.11 (Persistence of modes). *A* (*fault*) *mode* $m_{ji}(C_i)$ *is called persistent if for all temporal diagnoses* $\{(MA_k, I_k)\}$

$$\exists k' \; m_{ji}(C_i) \in MA_{k'} \quad \Rightarrow \quad \forall k > k' \; m_{ji}(C_i) \in MA_k$$

must hold.

It is called persistent in I_ω *if*

$$\forall k \; m_{ji}(C_i) \in MA_k.$$

We prefer this definition over the one proposed by [60] who calls a behavior persistent if the output of a component is a function of its inputs (and not of time), for two fundamental reasons: Firstly, persistence is a property of a **mode**, rather than of **model** as in [60]. Continuous models that reflect noise and uncertainty often cannot be stated in terms of functions, and a qualitative model that is obtained by abstracting a real-valued function is usually no longer a function. Secondly, there are only few kinds of systems that can be modeled in a directional way.

In contrast, the condition concerning the commonality of *sequence-constraints* is a property of the model.

Definition 10.12 (Homogeneity). *A model library LIB is called homogeneous, if there exists a set* sequence-constraints *that links states of the system at different time points and is shared by all models, i.e. for all modes* m_{ji},

$$\mathit{model}_{ji} = \mathit{state\text{-}constraints}_{ji} \cup \mathit{sequence\text{-}constraints},$$

and $\mathit{state\text{-}constraints}_{ji}$ *contains only restrictions for each single time point.*

If the above properties hold, then for persistent faults, being a state-based diagnosis can be not only a necessary condition for a temporal diagnosis (Lemma 10.1) but also a sufficient one [71], i.e. *MH* is a temporal diagnosis for *SD* and *OBSH* if and only if

$$MH = \{(MA, I_\omega)\}$$

and *MA* is a state-based diagnosis for *SD* and *OBSH*. Since the *sequence-constraints* do not contribute to a consistency check at a single time point, *MA* is obtained as a diagnosis for *SD sequence-constraints* and all OBS_i.

The above considerations apply, in particular, if all modeled behaviors are continuous. However, if the model contains discrete states and transitions between them, then homogeneity is usually violated, because the possible transitions are specific to a particular behavior mode. For instance, transitions between states *OPEN* and *CLOSED* do occur in the *OK* model, but not for a *STUCK* mode.

Even more fundamentally, the homogeneity property becomes obsolete, if the persistence assumption is dropped. So far, we considered only models that describe the component (system) **behavior** under each **mode** (assignment). In temporal diagnosis, we may want or need to model also, in which ways **mode changes** occur. Most attempts to do so make the assumption that this happens as a discrete change. Thus, the evolution of system behavior may be described by state changes within a mode and mode changes:

$$\begin{aligned}\mathit{sequence\text{-}constraints} = {} & \mathit{states\text{-}sequence\text{-}constraints}\\ & \cup \mathit{modes\text{-}sequence\text{-}constraints},\end{aligned}$$

although many formalisms represent them in the same way, namely as discrete transitions.

If we make a Markov assumption, then *sequence-constraints* become *transition-constraints*, which restrict pairs of adjacent states.

Transition-based diagnosis

There are a number of approaches to incorporating transitions to fault modes in the model, covering the spectrum from discrete-event models to models of continuous behavior. As basing such models on the concepts of states and transitions is natural, most of them are some variant of finite state machines or similar formalisms. There are many approaches, reflecting different types of systems, tasks, observations, temporal information, etc. Here, we can only provide a preliminary formal account for the common underlying ideas and refer to some specific instances. We represent a model of the possible evolution of the behavior a component, C_i, as a tuple

$$\mathit{model}(C_i) = (S_i, s_{i,0}, E_i, E_{i,obs}, T_i, T_{i,F})$$

with

- a finite set of **states**, S_i, which can represent operating modes under normal behavior (e.g., a proper valve in its closed state) or faulty behavior (the valve stuck closed),
- an **initial state**, $s_{i,0}$,
- a finite set of **events**, E_i, which may be exogenous influences, control commands (external or internal ones), the occurrence of faults, alarms or other observables, etc.,
- the **observable events**, $E_{i,obs} \subset E_i$, which exclude, at least, the events that trigger fault transitions (otherwise, there is no diagnostic problem),
- a finite set of **transitions**, T_i, shifting the system from one state to the next (deterministically or non-deterministically), based on the triggering event and possibly generating an event:

  $$T_i \subset S_i \times E_i \times S_i \times E_i.$$

  A transition $t \in T_i$ may represent switches between operating modes, shifts to a fault mode, but possibly also the return to a correct behavior in case of an intermittent fault or due to some repair or reset action,
- the set of **fault transitions**, $T_{i,F} \subset T_i$, which correspond to the occurrence of faults.

Such a model uses only the simplest representation of time, namely a (partial) ordering of states. A formal specification of the semantics can be based on some temporal logic containing a *next* operator [32]. Sometimes, metric temporal information (numerical or qualitative) may be necessary and/or available.

Such component models fulfill the important requirement to be **compositional**. We obtain a model of a system comprising a set of concurrently active components $\mathit{COMPS} = \{C_i\}$

$$\mathit{MODEL} = (S, s_0, E, E_{obs}, T, T_F)$$

as a product of the component models, where

$$S = S_1 \times \cdots \times S_n,$$

$$s_0 = (s_{1,0}, \ldots, s_{n,0}),$$

$$E = P\Big(\bigcup E_i\Big),$$
$$E_{obs} = P\Big(\bigcup E_{i,obs}\Big)$$

(which means, only part of the composite event may be observable). The way transitions are specified may depend on different assumptions, especially about synchronization of the local transitions.

A diagnosis is then, intuitively, some explanation of a sequence of observed events in terms of a path through the finite state machine which generates this observable trace and can be defined as follows.

Definition 10.13 (Transition-based diagnosis). *Let*

$$MODEL = (S, s_0, E, E_{obs}, T, T_F)$$

be the model of a system and

$$OBS = (OBS_1, \dots, OBS_n) \in E^n_{obs}$$

be a sequence of observations.

A sequence of events

$$e = (e_1, \dots, e_m) \in E^m$$

is an extension *of OBS, iff it contains OBS in the proper order, i.e.*

(i) $\forall k\ \exists j(k)\ e_{j(k)} \cap \bigcup_i E_{i,obs} = OBS_k$,

(ii) $k_1 < k_2 \ \Rightarrow\ j(k_1) < j(k_2)$;

e is a transition-based diagnosis *of* (*MODEL*, *OBS*) *iff*

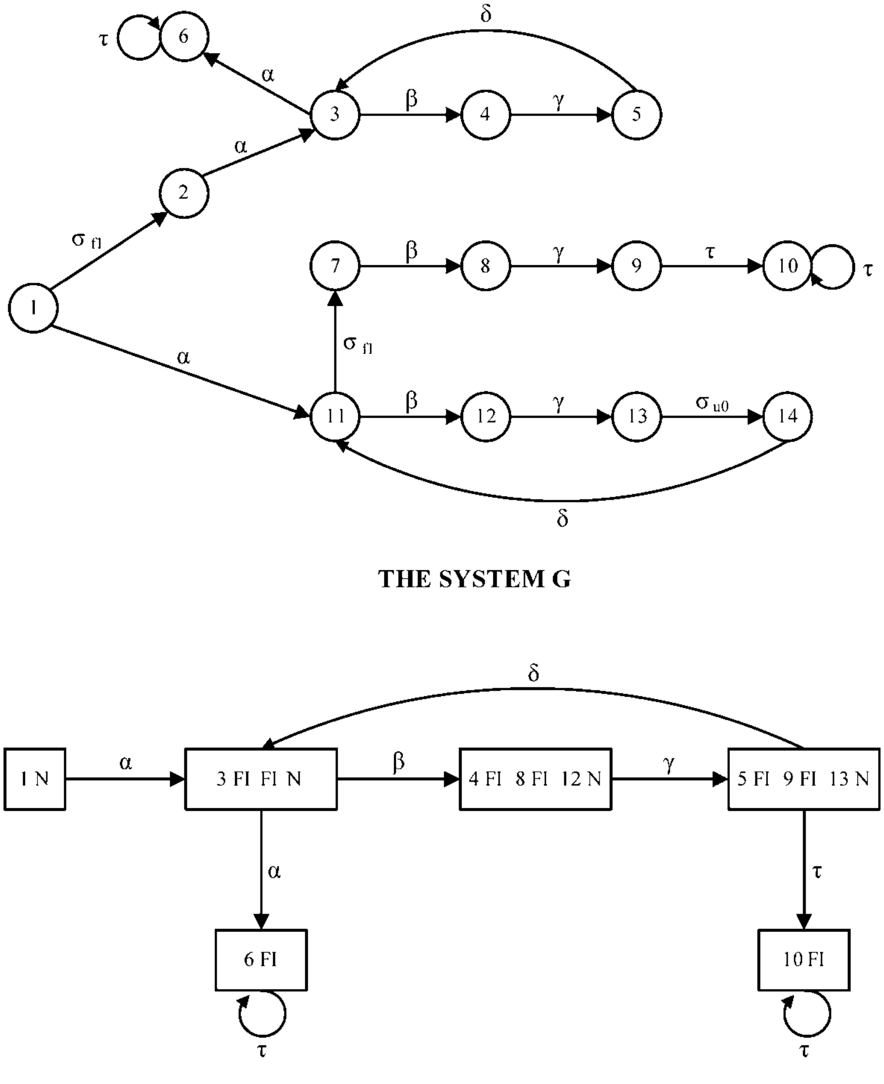
(i) *e is an extension of OBS*,

(ii) $\exists (s_1, \dots, s_m) \in S^m\ \forall 1 < j < m\ (s_{j-1}, e_j, s_j, e_{j+1}) \in T$.

Fault detection is performed, if every diagnosis contains some fault transition. Fault identification corresponds to the subsequence of fault transitions in a diagnosis, and fault localization is done by looking for the components where these fault transitions occur. Like for the snapshot case, we can apply some minimality criteria for ranking diagnoses and fault localizations. One could also recast the definition in terms of fault events or fault states (in the latter case with a modified minimality criterion). There are many directions for variations, specializations, and extensions of this perspective on diagnosis across time.

Diagnosis with discrete-event models

[65] uses a deterministic finite state machine without emitted events and approaches the diagnostic problem by compiling the original finite state machine into one that contains only observable transitions and produces the same language in terms of observations, called *diagnoser*. Its states represent the respective sets of nodes in the original model that can be reached via paths that contain also unobservable transitions,



THE SYSTEM G

THE DIAGNOSER G_d

Figure 10.4: The system model (top) with observable transitions α, β, γ, δ, τ and the fault transition σ_{Fl}. The nodes of the diagnoser contain the potentially reached original states together with the faults on the path ("Fl") or "N" (i.e. no fault). From [65].

labeled with the fault transitions on these paths. This means that, after a sequence of observations, the diagnoses can be read off of these labels (together with a prediction of the possible current states of the system). Fig. 10.4 gives a simple example of a finite state machine and its diagnoser.

Related work is described in [46] and [47]. [77] discusses links between this approach and diagnosability based on continuous models. When a system comprises many components that operate concurrently, the explosion of the Cartesian product of the states is an obvious problem and motivates a decentralized approach as in [54] applied to telecommunication networks.

Diagnosis with hybrid models

The view on a system's behavior as a sequence of state transitions lends itself to modeling various kinds of systems including combinations of software and physical components. In this case, the (continuous) behavior during a state may have to be modeled, as well, because it can be the cause of discrete changes. In *Livingstone* [84], components are also modeled as a graph of transitions between states of the component, which represent normal operating modes and fault modes. A component's behavior is characterized by a set of variables (physical quantities, commands, etc.). States are characterized by constraints on these variables, actually the assignment of a single value to some of the variables. As for the models of many of the consistency-based diagnosis systems discussed earlier in this section, qualitative modeling [81, 35, 4] turns the representation of the continuous behavior into a finite one (e.g., in terms of finite constraints or propositional logic).

A *Livingstone* model can be interpreted in the general framework outlined above in the following way. A component C_i has an associated vector of variables $\underline{v}_i$ with the finite domain $\mathrm{DOM}(\underline{v}_i)$. The behaviors under the different states $s_{ij} \in S_i$ are specified by constraints:

$$s_{ij} \Rightarrow C_{ij} \quad \text{with } C_{ij} \subset \mathrm{DOM}(\underline{v}_i).$$

Events E_{ij} are also specified in terms of constraints on the variables:

$$E_{ij} \subset \mathrm{DOM}(\underline{v}_i)$$

and transitions move from states that fulfill the triggering conditions to states that satisfy the resulting condition. In each state, besides a nominal transition, also a number of fault transitions can occur non-deterministically. Again, the entire model can be understood as split into a set of *state-constraints* and *transition-constraints*, with some non-determinism concerning the latter.

In order to form a system model, the composition of such component models happens at two levels. On the one hand, as usual, the interaction of components along the system structure is represented by shared variables between component models. They may correspond to physical quantities, such as pressure and flow, commands of a controller, etc. On the other hand, states, events, and transitions are aggregated (in a synchronous way).

Because events are specified as restrictions on variables, the observable ones correspond directly to the snapshot observations (e.g., measurement of a set of variables) discussed earlier. In contrast to the compilation of the transition system into the diagnoser, *Livingstone* generates diagnoses incrementally from snapshot to snapshot. Because of the combinatorics of multiple transitions from each local state, complete generation of all potential successor states is prohibitive for interesting applications. Like *SHERLOCK*, *Livingstone* focuses on tracking the most likely paths, exploiting the a posteriori probabilities of the transitions given the observations about the resulting state. The system and its predecessors were applied prototypically to spacecraft self-diagnosis as a basis for self-reconfiguration [84].

10.4.5 Abductive Diagnosis

The concept of diagnosis, so far, is based on finding system models that do not contradict the given observations. This may seem quite weak. In fact, if the system shows

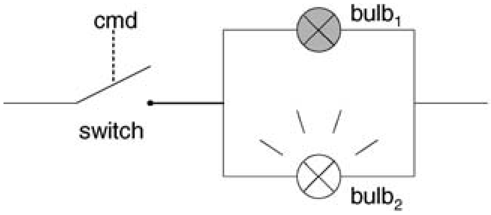


Figure 10.5: A fragment of a circuit with two parallel light bulbs.

some symptoms, we may want a diagnosis that provides a **causal account** for them. This idea leads to a new logical definition of a diagnosis, which requires that a model logically **entails** the given observations, rather than simply being consistent with them:

$$MODEL \models OBS$$

[57]. However, we have to be cautious when using this definition of *abductive diagnosis*. For instance, if the observations include a command "CLOSE" to the switch in the fragment of a circuit shown in Fig. 10.5, but bulb$_1$ remains dark while bulb$_2$ is lit, then the single fault in bulb$_1$ explains the observations of the bulbs, but we do not expect it could provide a reason for $switch.cmd = CLOSE$. Intuitively, we want the system response (the "output" variables) to be entailed, but not the exogenous features (the "input" or independent variables). The most general definition reflecting this intention is the following.

Definition 10.14 (Abductive diagnosis). *Let $MODEL_F$ be a model of a fault and $OBS = OBS_C \cup OBS_A$ be a set of observations. $MODEL_F$ is an abductive diagnosis iff it is consistent with OBS_C and entails OBS_A:*

$$MODEL_F \cup OBS_C \not\models \bot,$$
$$MODEL_F \cup OBS_C \models OBS_A.$$

A complete mode assignment MA is a component-oriented abductive diagnosis for a system description SD and a set of observations OBS iff

$$SD \cup \{MA\} \cup OBS_C \not\models \bot,$$
$$SD \cup \{MA\} \cup OBS_C \models OBS_A.$$

Please, note that if OBS_C refers to exogenous variables, the first condition is satisfied by any valid model (as discussed in Section 10.3). Console and Torasso [9] discuss the consequences of different possibilities to specify OBS_C and OBS_A. In our example, we would choose

$$OBS_C = \{\text{switch.cmd} = \text{CLOSE}\},$$
$$OBS_A = \{\text{bulb}_1\text{.light} = \text{OFF}, \text{bulb}_2\text{.light} = \text{ON}\}.$$

Unfortunately, a single fault in bulb$_1$ does not entail OBS_A based on OBS_C, because there is no information about the voltage supply and is not found as an abductive diagnosis, unless also the voltage supply is abduced.

Poole [57] raises the issue of how to represent the observations. Rather than treating them as a conjunction of inputs and outputs, we could try to find an explanation for observations stating that the input implies the output. This means in our example, we use

$$OBS = \{\text{switch.cmd} = \text{CLOSE} \Rightarrow (\text{bulb}_1.\text{light} = \text{OFF} \wedge \text{bulb}_2.\text{light} = \text{ON})\}$$

which would have to be entailed by an abductive diagnosis (which is again not the case). Note that we (humans) can even obtain a diagnosis solely based on observation of the outputs:

$$OBS = \{\text{bulb}_1.\text{light} = \text{OFF}, \text{bulb}_2.\text{light} = \text{ON}\}$$

and that consistency-based diagnosis with fault models produces the proper result.

Abductive diagnosis is attractive, because it provides a stronger notion of diagnosis which seems to reflect the aspect of causality in our human conception of diagnosis. However, apart from the fact that logical entailment is generally unrelated to causality, this stronger notion of diagnosis imposes stronger requirements on the model and the possible inferences, as illustrated by the above example. When compared to consistency-based diagnosis, the results are more sensitive to the particular representation and strength of the model and the observations. If an observation states that, say, a flow at some point is positive, while a model can only predict a disjunction $\mathit{flow} = \mathit{zero} \vee \mathit{flow} = \mathit{positive}$ (e.g., based on the model of a check valve), it would not be an abductive diagnosis. If the domain of flow (both in the model and the observation) would contain only the values *negative* and *non-negative*, then this would yield an abductive diagnosis. However, this coarser domain may then be too weak to derive some other predictions.

Of course, consistency-based diagnosis depends on the strength of the model, as well, and, in particular, on the granularity of the domains. This is because this can render the model unable to detect some of the existing conflicts. However, it is still guaranteed that the correct diagnosis (as a mode assignment) is never excluded. Such a guarantee cannot be obtained for abductive diagnosis.

Depending on the available observations, an abductive diagnosis may include not only the modes, but also the current state of the system and even numerical parameters (as suggested by [57]) which makes the abduction task even harder for systems of an interesting kind and size.

Abductive diagnosis seems to become feasible and provide some basis for meeting our intuition behind an explanation, if the model has causal notion embedded (as opposed to purely constraint- or equation-based behavior descriptions). In fact, many of the examples used for explaining abductive techniques come as causal networks that explicitly link faults to effects. As already discussed in Section 10.3, this kind of system-specific diagnosis task compiled into a system model is a non-solution to diagnostic applications (although in engineering practice, something similar is done in constructing fault trees for safety analysis), because it violates genericity, compositionality, and reusability of the model. What is required is a modeling ontology that captures causality and is compositional. As stated earlier, process-oriented modeling [34, 35, 41] is a candidate. This also paves a way to overcome some of the restrictive assumptions and limitations of component-oriented diagnosis discussed in Section 10.4.1.

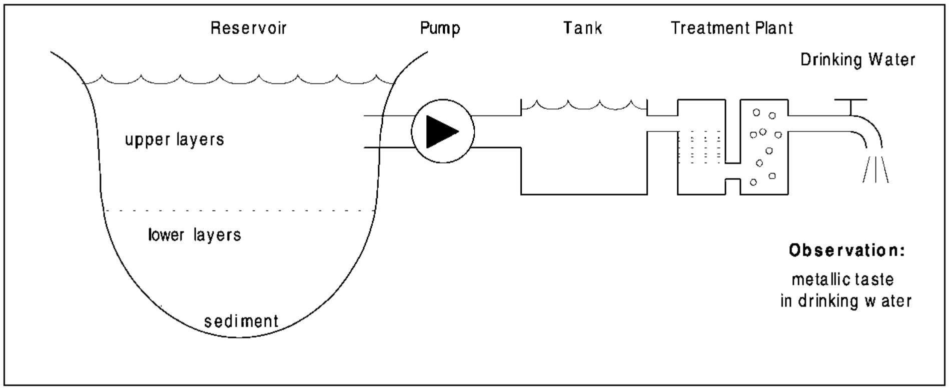


Figure 10.6: The drinking water has a high iron concentration, because solid iron in the sediment was re-dissolved into the water and transported to the tap.

10.4.6 Process-Oriented Diagnosis

In a simplified (though real) scenario of drinking water treatment (Fig. 10.6) [41], a high concentration of dissolved iron is detected in the drinking water. Since it exceeds the legally allowed level, and there is no source of iron the operators are aware of, this is a challenge for diagnosis. Human analysis yields to the following result: there is solid iron in the sediment of the reservoir, which was not known before. When the pH of the lower water layer, which is usually neutral, became acidic (most likely caused by some algal bloom phenomenon), this started a chemical process of redissolving of iron into the water body. The dissolved iron ascended to the surface layers, was captured with the raw water intake, and the treatment process did not reduce the unexpectedly high concentrations of iron as required. While we would claim that the case is clearly a diagnostic task, let us revisit the assumptions underlying component-oriented diagnosis as discussed in Section 10.4.1.

- **Fixed, well-specified set of components**: although there are components, such as pumps, containers, etc., the relevant diagnostic reasoning refers to biological, chemical, and physical processes; it would not be convenient to consider algae, water layers, etc. as components, and even if we do, solid iron was not a known "component" of the system.

- **Known, fixed structure**: the system and its model does not have a static structure; rather, there is a dynamically changing pattern of active processes and objects, substances, etc. appearing and disappearing.

- **Well-designed system**: although the treatment plant itself could be considered as such, the notion makes no sense for the reservoir and the processes involved. The *GOALS* are external to the model, and without making them explicit, there is no inconsistency.

- **Component faults only**: obviously, algae are not a fault, even if their biomass grows exponentially in an algal bloom period, nor is redissolving of iron under acidic conditions a fault mode of something; it is simply natural. But it is unwanted from the perspective of the violated *GOALS*.

- **No structural faults**: the nature of the disturbance **is** a structural change in the system, triggering unforeseen processes.
- **“Crisp” faults**: even more than with respect to artifacts, healthiness of ecological and biological systems, but also process plants is often expressed in terms of a spectrum of degradation, rather than a qualitative behavior change. Of course, in our example, the legal restrictions make it crisp.

Obviously, addressing such a diagnosis task in theory and implementation requires a different formalization of modeling and the diagnosis task. However, what we stick with is the idea that the answer to the diagnosis task is given by a model that is “compliant” with the observations of system behavior.

The example suggests the use of *process-oriented modeling*. Collins [7] develops the *Process Diagnosis Engine (PDE)* as abductive diagnosis on such a model. Heller and Struss [41] present a theory of process-oriented consistency-based diagnosis, realized as the *Generalized Diagnosis Engine* (G^+DE).

In a nutshell (see [34, 35, 41] for details), a process is considered as some elementary phenomenon, which can be modeled independently of others and is, therefore, suited to compositional modeling. This has two consequences: one is that a process model has to state explicitly all preconditions for the process to occur by listing the (typed) objects that interact in a particular configuration (*structural-conditions*) and constraints on involved quantities (*quantity-conditions*). A process can create new objects and relations between them (*structural-effects*) and affects quantities of the participating objects (*quantity-effects*). The second consequence of the required compositionality is that *quantity-effects* cannot all be simply stated as constraints on the quantities. This is because each (type of) process can only state a *partial* contribution to some overall effect. For instance, the model of the iron-redissolving process can only state that it adds to the concentration of dissolved iron in the water layer, but it cannot claim that this concentration effectively increases, because in a particular scenario, there may be other, counteracting, processes active that override the effect (e.g., oxidation of iron). In response to this, process-oriented modeling involves the concept of *influences* that goes beyond mathematical modeling based on (differential) equations, constraints, or first order logic. If some variable x influences a variable y, say, positively, written $I^+(x, y)$, this means basically that the derivative of y is a monotonic function of x:

$$I^+(x, y) \quad \Leftrightarrow \quad \exists f \; \frac{dy}{dt} = f(\ldots, x, \ldots) \wedge \frac{df}{dx} > 0.$$

The actual value of $\frac{dy}{dt}$ can only be determined after **all existing influences** can be (e.g., linearly) combined. But this requires a *closed-world assumption*, which provides an important hook for model revision during the search for a consistent model.

Thus, a process implies the effects, if the conditions are true:

$$\textit{structural-conditions} \wedge \textit{quantity-conditions}$$
$$\Rightarrow \textit{structural-effects} \wedge \textit{quantity-effects}$$

How can we state the diagnosis problem, which we call *situation assessment*, because there is not necessarily “something wrong”? We start with a partial description of a scenario in terms of objects, object relations, and variable values. This may include real observations, *OBS* (e.g., measurements, such as "iron concentration above threshold") and assumptions, *ASSM* (e.g., assertions that usually hold, such as "pH neutral"). The target is to construct process models that, based on propositions about structure (objects and object relations) and quantities, are consistent with *OBS* and, if possible, with *ASSM*. Again, we apply Occam's razor and prefer models that satisfy some minimality criteria. There are two orthogonal dimensions:

- do not drop more assumptions than required to obtain a consistent model,
- do not introduce more unanticipated objects than necessary in order to derive an explanation (why assume both solid iron in the sediment **and** iron in an affluent to the reservoir?): the structural basis should not be larger than necessary.

We should be more precise about the latter criterion: what we would like to minimize is the set of objects in the model that are not generated by some process, but are introduced without further justification by the model. In our example, the dissolved iron is an effect of the redissolving process, whereas the solid iron in the sediment does not have an explanation in the model. Hence, the issue relates to the question of the model boundaries: where should we stop to ask for reasons, because they are beyond what is captured by the model library? In our case, iron in the tank requires a causal explanation, whereas the existence of solid iron or algae does not. We characterize those types of object that sit "on the boundary" of the model as *introducible*. Obviously, they comprise those that never occur as a structural effect of a process in the library. But we may want to regards additional ones as introducible for certain problems or scenarios.

Based on this, we can give an informal definition of a process-oriented diagnosis (a formal account using default logic is described in [40]).

Definition 10.15 (Situation assessment). *Let LIB be a process library, OBJ-INTRO the set of introducible objects, OBJ-OBS and OBJ-ASSM the objects mentioned in OBS and ASSM, respectively. A situation assessment for* (*LIB*, *OBS*, *ASSM*) *is a triple* (*STRUCTURE*, *QUANT*, *ASSM-RETR*); *where STRUCTURE is a set of objects, and relations, QUANT is a set of value assignments to quantities, and ASSM-RETR* $\subset$ *ASSM a set of assumptions such that the resulting model is consistent with the observations and a subset of the assumptions*:

(i) $\mathit{STRUCTURE} \cup \mathit{QUANT} \cup \mathit{LIB} \cup \mathit{OBS} \cup (\mathit{ASSM} \setminus \mathit{ASSM\text{-}RETR}) \not\models \perp$

The structure contains the observed objects and a subset of the assumed ones:

(ii) $\mathit{OBJ\text{-}OBS} \subset \mathit{STRUCTURE}$, $\mathit{OBJ\text{-}ASSM} \setminus \mathit{ASSM\text{-}RETR} \subset \mathit{STRUCTURE}$,

(iii) *the model contains exactly the objects that are entailed by the introducible objects, i.e. the introducibles themselves and the ones created by processes*,

(iv) ($\mathit{STRUCTURE} \cap \mathit{OBJ\text{-}INTRO}$) $\cup$ *ASSM-RETR is a* (*with respect to set inclusion*) *minimal set that satisfies* (i) *through* (iii).

Different applications may require modifications to this definition which minimizes the set of newly introduced objects and retracted assumptions, while introduced

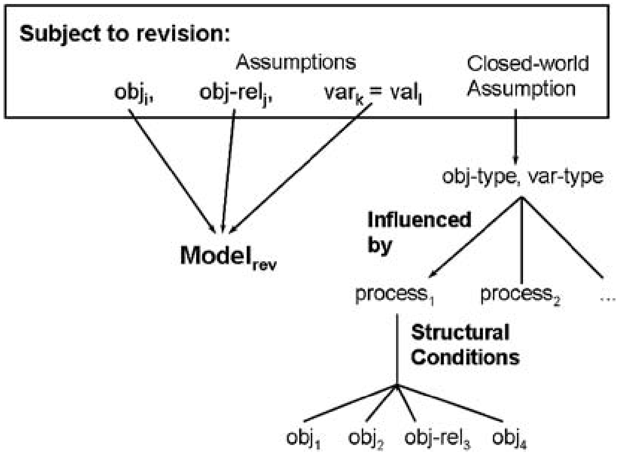


Figure 10.7: The options for model revisions.

relations do not count. Also, the likelihood of the occurrence of objects may further constrain the focus.

Regarding the implementation of a process-oriented diagnosis system, condition (i) suggests that algorithms from component-oriented consistency-based diagnosis can be applied: conflicts can be generated that contain user assumptions from *ASSM* and the closed world assumptions underlying the influence resolution for quantities. We point out that the latter introduce an element of abductive diagnosis: influence resolution implies that the derivative of a variable is zero if there is no influence acting on this variable. Therefore, if a change in a variable is observed (or postulated), any model that contains no process influencing this variable will be inconsistent, and the closed-world assumption for this variable occurs in a conflict. The same happens, of course, if the known influences contradict the (direction of) change in a variable. A similar technique can be used, if the *deviation* of some variable from an expected value is observed and the model captures how such deviations can emerge and propagate through a system. Resolving such conflicts and revising the model in a search process starts from the retraction of these closed-world assumptions. While in component-oriented diagnosis retraction of a mode assumption means switching to a different mode, retraction of the closed-world assumption requires to find (additional) potential influences on this variable. The search space for this revision is given by the process library: it contains a finite set of processes that can possibly influence a variable of the respective type associated with an object of the respective type. Extending the model by such a process may lead to a new inconsistency, if its quantity conditions are not satisfied. Also, the structural conditions need to be satisfied, and if they contain objects that are not yet included in the model, then condition (iii) requires that they either be introducible or explained by the structural effect of yet another process to be included. Also for this revision, the processes in the library can be searched for the appropriate candidates. Fig. 10.7 illustrates the search process. Compared to the component-oriented best-first search algorithms, the minimality criterion is less effective in this case, because it relates to the **ultimate** cause (the introducible objects), and a one-step look-ahead will not help.

The existing approaches along these lines (*PDE* [7], G^+DE [41]) are performing diagnosis in **one snapshot**. This is a serious limitation for many relevant diagnosis problems, since the origin of some disturbance may already have ceased to exist, while the effects persist. For instance, if we expect to detect a cause for the deviation in the pH (e.g., algal bloom), the actual observation may state that there is none and render such an explanation inconsistent.

Including the temporal dimension adds to the complexity issues of this approach and, together with the demand for good search heuristics makes it a real challenge to model-based diagnosis research. Any progress would contribute to a significant extension of the application scope of model-based diagnosis.

10.4.7 Model-based Diagnosis in Control Engineering

There exists another research area also called “model-based fault diagnosis and isolation”. It has emerged in control engineering, and, while sharing some basic commonalities with model-based diagnosis in Artificial Intelligence, involves quite different techniques. The common idea is to start diagnosis from the deviation of an observed behavior from a model of correct behavior and to view a diagnostic hypothesis as a model revision that removes this deviation. However, the techniques are purely mathematical, and the models used are usually numerical, non-compositional black-box models with a fixed (mathematical) structure, lacking an explicit conceptual layer of modeling and, hence, any symbolic reasoning and inferences. Partly, this reflects the application domain of process control and the kind of models used for this purpose. As a consequence, the kinds of faults that can be handled are limited to those that can be expressed as a variation of the mathematical OK-model (e.g., parameter deviations). Faults that modify the causal structure of the system and/or its mathematical structure constitute a problem, as opposed to the model-based methods described in this chapter. There are several attempts to compare, relate, and combine the different types of model-based diagnosis [18, 42, 1, 53].

10.5 Test and Measurement Proposal, Diagnosability Analysis

Usually, a diagnosis based on some initial set of observations does not yield a unique diagnosis result, even under certain preference criteria, such as minimality or likelihood. If the model has been fully applied and cannot provide more diagnostic information, the only source for further discrimination between the remaining diagnostic hypotheses is additional observations of the system behavior. This means observing additional variables and/or performing observations of the system in a different state or with different input. Therefore, the test generation task can be stated as determining which influences on the system and which observables promise information that refutes some of the current (diagnostic) hypotheses. A variant of this task is end-of-line testing, i.e. performing tests of a manufactured product that are suited to confirm that the product is not faulted. This may seem to be a different task, but it can only be achieved by tests that are designed to refute all possible faults (since this is not feasible in reality a set of plausible faults has to be selected, e.g., single faults, or the most probable ones). There is no way to confirm the presence of a particular behavior other than refuting all competing behavior hypotheses.

10.5.1 Test Generation

The core problem is to determine tests for discriminating between two possible behaviors of a system, i.e. two models. A test has to specify

- how to stimulate the system and
- what to observe of the system's response to this stimulus

in order to gain discriminating information. This requires fixing the possibilities of influencing the system, called *test inputs* or *stimuli* in the following, and the potential observations, *OBS*.

In the most general way, testing aims at finding out which model hypothesis out of a set *Hyp* is correct (if any) by stimulating a system such that the available observations of the system responses to the stimuli refute all but one hypotheses (or even all of them). This is captured by the following definition.

Definition 10.16 (Discriminating test input). *Let*

$TI = \{ti\}$ *be the set of possible test inputs* (*stimuli*),

$OBS = \{obs\}$ *the set of possible observations* (*system responses*), *and*

$Hyp = \{model_i\}$ *a set of hypotheses.*

*$ti \in TI$ is called a **definitely discriminating** test input for Hyp if*

(i) $\forall model_i \in Hyp\ \exists obs \in OBS,\ ti \wedge model_i \wedge obs \not\models \perp$

and

(ii) $\forall model_i \in Hyp \quad \forall obs \in OBS$
if $ti \wedge model_i \wedge obs \not\models \perp$
then $\forall model_j \neq model_i,\ ti \wedge model_j \wedge obs \models \perp.$

*ti is a **possibly discriminating** test input if*

(ii′) $\forall model_i \in Hyp\ \exists obs \in OBS$ *such that*
$ti \wedge model_i \wedge obs \not\models \perp$
and $\forall model_j \neq model_i,\ ti \wedge model_j \wedge obs \models \perp.$

*ti is a **not discriminating** otherwise.*

In this definition, condition (i) expresses that there exists an observable system response for each hypothesis under the test input. It also implies that test inputs are consistent with all hypotheses, i.e., we are able to apply the stimulus, because it is causally independent of the hypotheses. Regarding the model, this corresponds to the requirement that it captures the behavior under each tuple in the Cartesian product of the domains of exogenous variables, as discussed in Section 10.3. Condition (ii) formulates the requirement that the resulting observation guarantees that at most one hypothesis will not be refuted, while (ii′) states that each hypothesis **may** generate an observation that refutes all others.

Usually, one stimulus is not enough to perform the discrimination task which motivates the following definition.

Definition 10.17 (Discriminating test input set). *$\{ti_k\} = TI' \subset TI$ is called a discriminating test input set for $Hyp = \{model_i\}$*

if $\forall model_i, model_j$ with $model_i \neq model_j$

$\exists ti_k \in TI'$

such that ti_k is a (definitely or possibly) discriminating test input

for $\{model_i, model_j\}$.

*It is called **definitely discriminating** if all ti_k have this property, and **possibly discriminating** otherwise. It is called **minimal** if it has no proper subset $TI'' \subset TI'$ which is discriminating.*

This defines what we would like to obtain. Actually computing solutions faces a different dimension of complexity compared to diagnosis. In diagnosis, **one observation** of the system behavior in **one situation** (or a sequence of such situations) is **given** and needs to be checked for consistency with various models. For test generation, the space of **all situations and observations** has to be searched in order to find some that are inconsistent with at least one of the models. Intuitively, one would like to identify the differences in the space of all possible behaviors under two or more models. In contrast to consistency-based diagnostic reasoning which happens at the conceptual level (e.g., component behavior modes in component-oriented diagnosis), test generation has to analyze the behavior model itself, unless we apply an algorithm that generates test inputs and then tests them for consistency with the models.

Test generation with relational models

In the following, we outline a fairly general approach that assumes that models are represented as relations over a set of variables, but whose underlying ideas might be adapted to other modeling formalisms. The approach covers models that are given by equations and implemented by constraints. It is assumed that test inputs and observations can be described as value assignments to system variables. If the system is modeled as an aggregate of components, the hypotheses to be tested are given by (usually single) faults of components. If $\underline{v}_{Ci}$ is the vector of variables local to a component C_i with a domain $\mathrm{DOM}(\underline{v}_{Ci})$, each possible behavior mode $mode_{ij}$ of C_i has an associated relation

$$R_{ij} \subseteq \mathrm{DOM}(\underline{v}_{Ci})$$

as a behavior model. A fault hypothesis in testing is then given by the join of the relations that correspond to a particular assignment of modes, MA, to the components:

$$R(MA) = \underset{mode_{ij} \in MA}{\bowtie} R_{ij}.$$

Once this relation is constructed, the component structure is no longer relevant. Hence, we can choose a more general relational representation which covers testing of arbitrary hypotheses that can be stated in terms of a set of interrelated variables. This includes tests that aim at identifying a state variable which is not directly observable, testing applied to systems that are modeled in a process-oriented formalism, and the design of experiments for checking different modeling hypotheses. Thus, the system behavior is assumed to be characterized by a vector

$$\underline{v}_S = (v_1, v_2, v_3, \ldots, v_n)$$

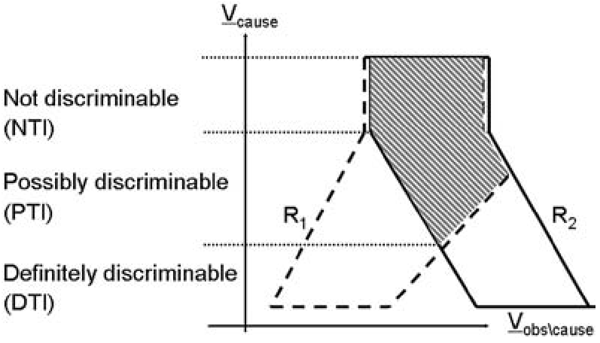


Figure 10.8: Discriminating inputs.

of system variables with domains

$$\mathrm{DOM}(\underline{v}_S) = \mathrm{DOM}(v_1) \times \mathrm{DOM}(v_2) \times \mathrm{DOM}(v_3) \times \cdots \times \mathrm{DOM}(v_n).$$

Then a hypothesis $model_i \in Hyp$ is given as a relation

$$R_i \subseteq \mathrm{DOM}(\underline{v}_S).$$

Observations are value assignments to a subvector of the variables, $\underline{v}_{obs}$, and also the stimuli are described by assigning values to a vector $\underline{v}_{cause}$ of susceptible ("causal" or input) variables. We make the, not very restrictive, assumption that we always know the applied stimulus which means the causal variables are a subvector of the observable ones:

$$\underline{v}_{cause} \subseteq \underline{v}_{obs} \subseteq \{v_i\}.$$

The basic idea underlying test generation [70] is then that the construction of test inputs is done by computing them from the observable differences of the relations that represent the various hypotheses. Fig. 10.8 illustrates this. Firstly, for testing, only the observables matter. Accordingly, Fig. 10.8 depicts only the projections, $p_{obs}(R_1)$, $p_{obs}(R_2)$, of two relations, R_1 and R_2, (which are defined over a larger set of variables) to the observable variables. The vertical axis represents the causal variables, whereas the horizontal axis shows the other observable variables (which represent the observable response of the system). To construct a (definitely) discriminating test input, we have to avoid stimuli that can lead to the same observable system response for both relations, i.e. stimuli that may lead to an observation in the intersection

$$(p_{obs}(R_i) \cap p_{obs}(R_j))$$

shaded in Fig. 10.8. These test inputs we find by projecting the intersection to the causal variables:

$$p_{cause}(p_{obs}(R_i) \cap p_{obs}(R_j)).$$

The complement of this is the complete set of all test inputs that are guaranteed to produce different system responses under the two hypotheses:

$$DTI_{ij} = \mathrm{DOM}(\underline{v}_{cause}) \setminus p_{cause}(p_{obs}(R_i) \cap p_{obs}(R_j)).$$

Lemma 10.2. *If $model_i = R_i$, $model_j = R_j$, $TI = \mathrm{DOM}(\underline{v}_{cause})$, and $OBS = \mathrm{DOM}(\underline{v}_{obs})$, then DTI_{ij} is the set of all definitely discriminating test inputs for $\{model_i, model_j\}$.*

Please, note that we assume that the projections of R_i and R_j cover the entire domain of the causal variables which corresponds to condition (i) in the definition of the test input.

We only mention the fact, that, when applying tests in practice, one may have to avoid certain stimuli because they carry the risk of damaging or destroying the system or to create catastrophic effects as long as certain faults have not been ruled out. In this case, the admissible test inputs are given by some set $R_{adm} \subseteq \mathrm{DOM}(\underline{v}_{cause})$, and we obtain

$$DTI_{adm,ij} = R_{adm} \setminus p_{cause}(p_{obs}(R_i) \cap p_{obs}(R_j)).$$

In a similar way as DTI_{ij}, we can compute the set of test inputs that are guaranteed to create indistinguishable observable responses under both hypotheses, i.e. they cannot produce observations in the difference of the relations:

$$(p_{obs}(R_i) \setminus p_{obs}(R_j)) \cup (p_{obs}(R_i) \setminus p_{obs}(R_i)).$$

Then the non-discriminating test inputs are

$$\begin{aligned} NTI_{ij} = \mathrm{DOM}(\underline{v}_{cause}) \setminus p_{cause}((&p_{obs}(R_j) \setminus p_{obs}(R_i)) \\ &\cup (p_{obs}(R_i) \setminus p_{obs}(R_j))) \end{aligned}$$

All other test inputs may or may not lead to discrimination.

Lemma 10.3. *The set of all possibly discriminating test inputs for a pair of hypotheses $\{model_i, model_j\}$ is given by*

$$PTI_{ij} = \mathrm{DOM}(\underline{v}_{cause}) \setminus (NTI_{ij} \cup DTI_{ij}).$$

The $\frac{1}{2} * (n^2 - n)$ sets DTI_{ij} for all pairs $\{model_i, model_j\}$, $i < j$, provide the space for constructing (minimal) discriminating test input sets.

Lemma 10.4. *The (minimal) hitting sets of the set $\{DTI_{ij}\}$ are the (minimal) definitely discriminating test input sets.*

Note that Lemma 10.4 has only the purpose to characterize all discriminating test input sets. Since we need **only one** test input to perform the test, which can be computed in linear time, we are not bothered by the complexity of computing all hitting sets.

This way, the number of tests constructed can be less than $\frac{1}{2} * (n^2 - n)$. If the tests have a fixed cost associated, then the cheapest test set can be found among the minimal sets. However, it is worth noting that the test input sets are the minimal ones that **guarantee** the discrimination among the hypotheses in *Hyp*. In practice, only a subset of the tests may have to be executed, because some of them refute more hypotheses than guaranteed (because they are a *possibly discriminating* test for some other pair of hypotheses) and render other tests unnecessary.

Note that the required operations on the relations are applied to the **observable variables** only (including the causal variables). The projection of the entire relation R_i to this space is a step of compiling the composite model to one that directly relates the stimuli and the observable response. In some relevant applications, this space is predefined and small. For instance, when testing of car subsystems exploits the on-board actuators and sensors only, this may involve some 10–20 variables or so. The entire workshop diagnosis task has more potential probing points, but still involves only a small subset of the variables in the entire behavior relation R_i. Also note that this compiled model can be re-used for diagnosis purposes. Such a compact model may actually make the computation of the set $\{DTI_{ij}\}$ feasible if, for instance, finite relations representing qualitative models are used. [26] perform the computation on an ordered multiple decision diagrams OMDD representation. However, the compilation step can become expensive and practically infeasible. In cases where the complete computation of $\{DTI_{ij}\}$ is not possible, test generation can be done by search, and Lemmata 10.2 and 10.4 describe the search space.

[49] assumes a model stated in first order logic, which leads to a characterization of tests as prime implicants. In [80], sets of behaviors generated by qualitative simulation of competing models are used to search for discriminating experiments.

Although the **set** of discriminating test inputs given by Lemma 10.4 is minimal, the individual **test inputs** are not necessarily minimal in the sense that they are always specified by the entire given set of observables, despite the fact that only a subset of the stimuli and/or a subset of the other observables may suffice to produce the same discrimination effect. This is important for applications, since both the production of a stimulus and the performance of an observation are actions that determine the cost of testing, and justifies spending computation time on the reduction of test inputs. In [73], this is done by analysis and operations of the entire relations DTI_{ij}. Tests as prime implicants in the work of [49] already include this minimization step, but finding them is also exponential. But, again, under economic considerations, spending even days of computation time pays off, if it saves only seconds for an individual test that is carried out many times or if it allows workshop mechanics to avoid some expensive experiments in diagnosis.

If hypotheses are given as models of mode assignments, the set of definitely discriminating test inputs as defined above may be empty for two models, although discrimination may be possible through **several** tests. This may occur when there are internal states that cannot be observed or unambiguously inferred. As stated above, the approach can be used for state identification, as well, and the solution to the problem is to make hypotheses about the (relevant) states explicit as a set Hyp_{state} and include their determination in the testing. Since the new set of hypotheses becomes the Cartesian product

$$Hyp' = Hyp \times Hyp_{state}$$

this step increases the complexity of the task like the consideration of multiple faults does. Obviously, the solution is based on an assumption of persistence of states during testing.

Intrusive testing and probing

The solutions outlined above ignore or, a least, do not explicitly treat, an important feature of the real task in a practical context: unless we are using only pre-established sensors that are reflected in the system model, performing a test involves often much more than manipulating the input and/or state of the systems and some passive observation of its response. It may, temporarily or permanently, modify the structure of the system and, hence, the model. Even simply opening an electrical circuit and attaching a measurement device creates a new circuit. Other tests (e.g., in the medical domain) may even modify the system structure in an irreversible way. Hence, we do not only have to consider preparatory actions like removing a cover or lifting a vehicle, but modifications that affect the behavior of the system and, hence, have to be reflected by a change in the structure of its model. We will return to this issue in a broader context in Section 10.6.

10.5.2 Entropy-based Test Selection

Achieving optimality with respect to the cost during repeated use of a set of tests also requires to take into account the likelihood of different model hypotheses. This can be reflected in the sequence of tests or by dynamically choosing a new test based on the result of the previous one. If we assume that each hypothesis $model_i \in Hyp$ has a probability $p(model_i)$, then

$$H = -\sum_{model_i \in HYP} \big(p(model_i) \cdot \log\big(p(model_i)\big)\big)$$

is the entropy, a measure for the uncertainty in the information at this stage. In our context, it can be understood as an estimation of the number of tests to be performed in order to identify the true model. In the component-oriented case, the initial probabilities could be computed as the product of the a priori probabilities of the respective modes (under the condition that they are independent). When choosing the next test input, $ti \in TI$, we would like to maximize the expected information gain

$$H - H_e(ti),$$

where the entropy after applying ti has to be estimated over all observations that can possibly result from ti under the different hypotheses:

$$H_e(ti) = -\sum_{obs \in OBS(ti)} \Bigg(p(obs) \cdot \sum_{model_i \in HYP} \big(p(model_i \mid obs, ti) \cdot \log\big(p(model_i \mid obs, ti)\big)\big)\Bigg).$$

If a hypothesis $model_i$ is specified by a relation R_i, then

$$OBS(ti) = \bigcup_i p_{obs}(ti \bowtie R_i).$$

Finally, we include the possibility that the observation after a applying a test input under a hypothesis is not unique, but, instead, there is a probability distribution:

$$p(obs \mid model_i, ti).$$

After applying Bayes' rule and some transformations [70], we derive the following

Probabilistic test selection strategy

In order to discriminate among hypotheses $model_i \in Hyp$, choose a test input ti and a vector of observable variables $\underline{v}_{obs}$, such that

$$-\sum_{obs \in OBS(ti)} \bigl(p(obs \mid ti) \cdot \log\bigl(p(obs \mid ti)\bigr)\bigr)$$
$$+ \sum_{model_i \in HYP} \Bigl(p(model_i) \cdot \sum_{obs \in OBS(ti)} \bigl(p(obs \mid model_i, ti)$$
$$\cdot \log\bigl(p(obs \mid model_i, ti)\bigr)\bigr)\Bigr)$$

is maximal, where $obs \in \mathrm{DOM}(\underline{v}_{obs})$ and the probabilities of observations are determined from hypothesis-specific distributions:

$$p(obs \mid ti) = \sum_{model_i \in HYP} \bigl(p(obs \mid model_i, ti) \cdot p(model_i)\bigr).$$

There is an intuitive interpretation of the criterion used in the strategy: the first term is the entropy of the observations given the test input, which is maximal if they are equally distributed. The second term will be minimal if each model predicts unique values. Together, this meets our intuition that says a test is most informative if it leads to distinct values for the various hypotheses.

10.5.3 Probe Selection

The above strategy allows varying both, the input ti and the observable variables $\underline{v}_{obs}$. On the one hand, this includes the situation where the set of observables is fixed (e.g., by the existing on-board sensors of a space craft). On the other hand, applying stimuli to the system in order to gain diagnostically relevant information may not always be possible (e.g., in plants under continuous operation or in natural systems), but information may be obtained by measuring additional variables in the given situation. This task which appears as probe selection, measurement proposal, and sensor placement can be handled as a specialization of the above strategy, where the stimulus is fixed and an informative set of observable variables has to be determined.

It can be seen as a generalization of the probe selection strategy in *GDE* [20], which determines the best individual variable v_{obs} to be measured, based on the assumption that each hypothesis either implies a unique prediction of a value $obs_i \in \mathrm{DOM}(v_{obs})$ (defining the subsets HYP_i) or no prediction at all (the subset HYP_u), which is reasonable for the implementation based on value propagation and dependency recording. Furthermore, Kleer and Williams [20] use an equal distribution of values for the hypotheses in HYP_u, and, hence, estimates the probability of a measurement $obs_i \in \mathrm{DOM}(v_{obs})$ for v_{obs} as

$$p(obs_i) = p(HYP_i) + \frac{1}{m} \cdot p(HYP_u) \quad \text{where } m = \bigl|\mathrm{DOM}(v_{obs})\bigr|.$$

This transforms the criterion in the strategy to the expression

$$\sum_{obs_i \in \mathrm{DOM}(v_{obs})} \big(p(obs_i)\log\big(p(obs_i)\big)\big) + p(H_u)\cdot \log m$$

which should be minimized to obtain the best next measurement. One should note that even if no fault models are used, fault hypotheses do predict values based on the models of the non-faulty components. If, as in *GDE*, an *ATMS* is used for recording the dependency of predicted values on mode assignments this delivers the basis for determining the sets HYP_i and only the entropy computation has to be realized.

While probing helps, if the initial observations are not sufficiently discriminating, the probe selection strategy can also be beneficial in the opposite case, namely when there is an overwhelming amount of observations. [2] exploits it as a filter to extract relevant information from a message burst caused by a disturbance in a power distribution network.

10.5.4 Diagnosability Analysis

The question whether and how faults can be detected or discriminated from each other is relevant already during the design phase. If design for diagnosability and, in particular, placement of sensors for diagnostic purposes is a concern, variants of the techniques described above can be applied. Also failure-modes-and-effects analysis (FMEA) includes an analysis of the detectability of a fault. In this kind of analysis, the consideration is usually not on actively influencing the system. Rather, we expect **discriminability analysis** to answer the question *"For a particular design and a chosen set of sensors, determine whether and under which circumstances the considered (classes of) faults considered can be distinguished from each other (based on the sensor readings)"*. Fault detectability can be seen as a special case, namely the discrimination of faulty behaviors from the OK behavior.

Discriminability may depend on certain external conditions and internal states of the system. For instance, a certain fault of a particular sensor may only show up in a special temperature range, and a problem in the gear box may only affect driving in 2nd gear. If we replace the test inputs in the above definitions and algorithms by the set of such possible conditions, the techniques for test generation can be re-used. In [26], this is done using the relational behavior presentation. Detectability analysis has also been treated for discrete-event models, by analyzing whether a fault transition results in a visible trace different from OK behavior (within a certain number of transitions) (see [65, 77]).

10.6 Remedy Proposal

So far, all the tasks considered were focused on obtaining and using information in order to assess the behavior mode or state of systems, especially of systems whose behavior deviates from the normal and intended one. However, this is never a goal in itself, but only interesting as an input to some decision and action that requires this information. Diagnosis is only relevant if it supports a decision (whether and) how to re-establish the functionality of the misbehaving system, at least to a possible degree.

Actually, this purpose, which varies according to the type of system and the practical context of the task, ought to influence the nature of the expected diagnostic result and also the diagnostic process itself. For instance, on-board diagnostics for a vehicle subsystem should aim at the discrimination between classes of faults that, due to their nature and criticality, require different immediate recovery and safety actions, whereas off-board diagnosis of the same subsystem is focusing on discrimination between different suspect components in order to find the ones that need to be replaced. Usually, there is no need for continued discrimination if this does not influence the choice of the remedial action. Although this issue is both obvious and fundamental to diagnosis, it has been mainly ignored in theoretical work, and there are (too) few contributions to treating this means-end relationship in a general and systematic way [58].

In fact, in the context of real diagnosis work processes, the interdependency often becomes even tighter, bidirectional and more complex, because the respective activities become intermingled: (partial) repair actions may be carried out to support the overall diagnosis process. As pointed out earlier, the focus on fault localization in early work on diagnosis can be explained by an (implicit) focus on replacement of components as the remedial action. However, component replacement is but one special instance of actions for moving a system back to a healthy state and, in fact, impossible in some applications (e.g., space craft outside an orbit).

The diagnostic and testing theories and systems presented above are attempts to automate **reasoning** tasks, namely to infer diagnoses from observations and to propose informative observations based on the previous results. However, in particular in an industrial environment, in general, it is not these reasoning activities that are expensive, but efforts spent on **acting**, such as de-assembling a device, installing measurement equipment, and repairing the device. Compared to this, the time and cost spent on thinking is often negligible, and the result of this thinking matters only if it contributes to optimizing the overall workflow. The chance for diagnostic solutions to be really employed in practice is heavily reduced if they are not designed and developed under this perspective. It should be noted, though, that the above considerations apply only in a restricted way to on-board diagnostics, because they do not trigger directly expensive human activities.

These considerations motivate work aiming at model-based generation of proposals for remedies, at an integrated perspective on diagnosis, testing, and applying remedies, and at the integration of planning with model-based problem solving. Remedies can involve a whole range of different actions that need to be reflected in model-based systems in different:

- **replacement of components** that are suspect of failing usually leaves the structure of the device (and, hence, of the model) unchanged and simply changes the behavior mode (if successful); however, sometimes, a component may be replaced by one with different parameters or of a different type,

- **reconfiguration** exploits the structural redundancy of a device, which might achieve the specified purpose in different ways and even under fault conditions. Aircraft and space craft are equipped with redundant subsystems for critical functions, and power networks are huge networks of switches that enable the generation of different topologies with different paths between voltage sources and sinks; since the components that modify the topology (switches, valves, etc.)

are also components, the structure of the system (and of the model) remains unchanged, only the states of these components are affected,

- **modification of operating regions** is based on a system property that allows achieving certain goals with different settings of parameters and inputs; if one out of three heating elements in a room is not working, you may compensate for this by increasing the set point of the other two elements [79]; again, the structure of the device and the model remains the same,

- **modification of control** affects a special component, software; this step may correspond to implementing the previous two remedies, but it may also mean switching to a different control regime (e.g., from closed-loop to open-loop control in the case a sensor is suspect, or a control unit on a vehicle may replace an implausible value of one wheel speed sensor by some approximation gained from the other three sensors),

- **structural modifications** cover a wide range, from inserting new components (e.g., an electrical heating element) and establishing new connections (e.g., to bridge a series of electrical connectors, one of which is open, by a cable) to introducing ozone in a water treatment plant in order to trigger a process of oxidation of dissolved metals in the water; all this clearly results in a model that might be quite different from the designed or previous one. As stated above, also measurement actions may affect the structure of the system.

Some of these actions require continuing the analysis after their performance with a **new model**. But even those that do not, raise the question how they affect the state of the system, i.e., about the persistence of what has been observed or inferred before. What remains true after replacing a capacitor in a circuit? Some of the measured values may, others may not, and redoing all measurements may be a waste of efforts. (Immediately) after adding ozone to the water, the iron concentration is still the same, but its derivative is modified due to the oxidation process. This can be seen as an instance of problems connected to reasoning about actions (see, e.g., [45]), but the specific context (and the existence of a model) can offer special solutions.

10.6.1 Integration of Diagnosis and Remedy Actions

The discussion above shows that, rather than considering diagnosis in isolation (as in Section 10.4), we need to model a process that

- integrates actions of testing and therapy and the inference of diagnostic hypotheses based on the results of such actions,

- may change the system model dynamically,

- is guided by the goal of re-establishing the original or some weakened functionality of the system.

Thus, we have a task similar to diagnosis across time in the sense that a history of possible diagnoses has to be maintained and updated over time. The difference is that transitions may be due to actions, that they may affect the system structure, and that the intended function of the system has to be modeled and reasoned about. Producing a complete representation of all possible transitions, e.g., in terms of a finite state machine, appears feasible only if there are no significant structural changes included in the remedy actions, for instance, if only state changes, reset, or replacement actions are available.

[30] proposed a general formalization of such an integrated process for component-oriented diagnosis and repair, which also takes into account that actions may fail. Slightly modifying and simplifying their proposal (assuming actions are instantaneous and cannot fail), we obtain an extension of our Definition 10.8 (Temporal diagnosis). We introduce an *action history*

$$AH = \{(ACT_l, t_l)\}, \quad t_l \in I_\omega$$

and a *goal history*

$$GOALH = \{(GOAL_m, I_m)\}, \quad I_m \in I_\omega,$$

which allows us to express both ultimate and intermediate goals. For instance, during the reconfiguration of a power transportation network, one has to avoid overload of certain lines, and may also have to make sure that certain critical consumers are never temporarily cut off from power supply. For replacement and reconfiguration, actions modify the modes and states of components, and in the latter case, change the system topology within the limits determined by the redundancy in the original structure. While this leaves

$$SD = LIB \cup STRUCTURE$$

unmodified and stable as in component-oriented diagnosis, the structure may also be subject to modification by remedial (and also measurement) actions. In this case, an extended system description $\overline{SD}$ has to comprise constraints on admissible structures.

The task is then to find a sequence of actions that is consistent with or achieves *GOAL* or a set thereof.

Definition 10.18 (Remedy). *An action history AH is a consistency-based remedy for* $\overline{SD}$, *OBSH*, *GOALSH and a mode history*, *MH if*

$$\overline{SD} \cup MH \cup AH \cup OBSH \cup GOALSH \not\models \bot$$

and an abductive remedy if

$$\overline{SD} \cup MH \cup AH \cup OBSH \models GOALSH.$$

It is called a consistency-based (*abductive*) *remedy of mode histories*, $\{MH_i\}$ *if it is a remedy for each* MH_i.

The second part of the definition reflects the fact that one may want a remedy that is known to work even though there is no unique diagnosis.

Unless there is a pre-specified set of repair plans to choose from, a planner is needed to generate such plans, and probabilities and cost have to be considered when selecting some optimized plan. While [75] present a cost function for a process including measurement and replacement, [30] propose an estimation of costs of plans for their approach that also takes into account down time of the system, which is a major issue in several applications (e.g., power transportation systems).

10.6.2 Component-oriented Reconfiguration

The idea of consistency-based diagnosis can be extended in a natural way to address the reconfiguration problem (see, e.g., [8]). In diagnosis, we are searching for a (minimal) revision, *MA*, of the mode assignment MA_{OK} that is consistent with observations:

$$SD \cup \{MA\} \cup OBS \not\models \bot,$$

where $MA \setminus MA_{OK}$ is minimal.

In analogy, we can consider the reconfiguration problem as a search for a (minimal) revision of the actual states of the reconfigurable components that is consistent with the behavior specification of the system, *GOALS*. More precisely, we assume that there exists a subset $COMPS_R \subseteq COMPS$ of components that enable the modification of the system topology (i.e. the interaction paths among the components) through manipulation of their states. Typical examples of such components are electrical switches (e.g., breakers in a power network) and valves (e.g., in the propulsion system of a space craft). In addition, there may be other components that can be (de-)activated, such as power generators, pumps, etc.

To support reconfiguration, the diagnosis step has to produce not only consistent mode assignments, *MA*, but also information about the states of the reconfigurable components.

Definition 10.19 (State assignment). *Let* $COMPS'_R \subseteq COMPS_R \subseteq COMPS$ *be a subset of the reconfigurable components. Then*

$$\bigwedge_{C_i \in COMPS'_R} s_{i_j}(C_i), \quad \text{where } s_{i_j} \in states(C_i)$$

is a state assignment. It is called complete if $COMPS'_R = COMPS_R$.

A diagnosis, *MA*, and a consistent (actual) state assignment SA_A, i.e.

$$SD \cup \{MA\} \cup \{SA_A\} \cup OBS \not\models \bot$$

require reconfiguration if they are inconsistent with *GOALS*:

$$SD \cup \{MA\} \cup \{SA_A\} \cup OBS \cup GOALS \models \bot.$$

If a replacement, self-healing, or reset of the broken components is not possible (i.e. *MA* is fixed), reconfiguration looks for a different state assignment that removes the inconsistency. The attempt to capture this intuition in a rigorous way, is not as straightforward as it appears at a first glance. The reason for this lies in the fact that modifying the state assignment will also modify the values of variables (actually, that is the purpose) and render observed variables obsolete in the goal situation. Some observed values will persist, and their information may be essential for the achievement of the goal. For instance, modifying switch positions in the power network affects voltages and current on the connected lines, but not the output of the generators in the network, and the observation of the latter can be essential for determining an appropriate reconfiguration; after all, you do not want to connect a consumer to an inactive generator.

The problem is an instance of the frame problem that occurs in reasoning about action and time. A general solution would have to be based on inferences that implement the idea that “only those observations persist that are not forced to change by the reconfiguration”. There may be domain-specific solutions that are based on an a-priori classification of persistent and non-persistent types of observations, as indicated for the power network example. They could also be ontology-specific, as discussed in Section 10.6.3. The following definition assumes that the set of persistent observations, OBS_P, can be determined in some way.

Definition 10.20 (Consistency-based reconfiguration). *Let MA be a diagnosis of SD and OBS, $OBS_P \subset OBS$ its persistent subset, and SA_A be the actual state assignment such that*

$$SD \cup MA \cup SA_A \cup OBS \nvDash \bot.$$

A state assignment SA_G that is consistent with SD, MA, OBS_P and $GOALS$,

$$SD \cup MA \cup SA_G \cup OBS_P \cup GOALS \nvDash \bot$$

is called a (consistency-based) reconfiguration for MA.

It is called minimal with respect to SA_A, if

$$SA_G \setminus SA_A$$

is minimal with respect to set inclusion.

Let $\{MA_i\}$ be a set of diagnoses, and for each i_0, SA_{M,i_0} the maximal entailed (partial) state assignment:

$$SD \cup \{MA_{i_0}\} \cup OBS \vDash SA_{M,i_0}.$$

SA_G is called a reconfiguration for $\{MA_i\}$, if it is a reconfiguration for each MA_i.

It is called minimal, if

$$SA_G \setminus \bigcap_i SA_{M,i}$$

is minimal.

There is no guarantee that, for a given diagnosis MA, a reconfiguration actually exists. But it does exist if and only if

$$SD \cup \{MA\} \cup GOALS \nvDash \bot$$

(provided SD contains the domain axioms for the states).

As already stated earlier in a more general way, what is really wanted is a **guarantee** that the reconfiguration achieves the goals,

$$SD \cup \{MA_i\} \cup \{SA_G\} \vDash GOALS$$

rather than being merely **consistent** with them. With both definitions, we may encounter problems in case of insufficient observations, an incomplete predictor and consistency check, and a weak model. The latter case may occur, for example, due to the lack of expressiveness regarding causality. For instance, in a relational behavior model, without further constructs, the **observation** of voltage being present may not be distinguishable from stating the **goal** that voltage be present. The local model of an open power line or breaker does not restrict the voltage on either side and may, hence, be consistent with the goal of a voltage request of a consumer, if the causal aspect is not represented that there has to be a source connected to produce it.

Incomplete information

The second part (on sets of diagnoses) of Definition 10.18 reflects one important motivation for the integration of diagnosis and repair, namely to avoid spending more efforts on the diagnosis step (i.e. the identification of modes and states) than necessary to determine appropriate remedies. There may be competing possible diagnoses and limited information about the actual states of components, but a reconfiguration might exist that can be shown to achieve the (or some) functionality again. For instance, the messages transmitted to the operator of a power network will often not enable him to localize the shorted component unambiguously, but nevertheless allow him to re-establish power supply by a topology that does not rely on any of the suspect components and, actually, he has to, within a minute or so. Later, of course, before sending off the repair staff, one better determines the fault location as accurately as possible, which may require more detailed (numerical) data and analysis.

These considerations mainly apply in case obtaining more discriminative observations requires costly actions. In on-board diagnosis, the set of available observations is usually fixed and basically comes for free. As pointed out earlier, it may be case, though, that the amount of data is overwhelming (but highly redundant) and require computation for selecting the most informative bulk of data. In this case, which occurs, for instance, in the power network application, the techniques for probe selection (Section 10.5.3) can be exploited as a filter (see [2]).

Minimality and cost

The definition of a minimal reconfiguration captures the idea that a maximal number of reconfigurable components should maintain their actual states. There can be many reasons why this may not suffice to reflect practical requirements appropriately. First of all, to select the best reconfiguration, costs of (types of) reconfiguration actions have to be considered which may differ (e.g., changing a switch position vs. turning on a new generator). Also one might prefer reversible actions over irreversible ones (such as firing a pyro valve). Under the assumption that the cost of reconfiguration grows monotonically with the set of actions, the set of minimal reconfigurations contains the cheapest one(s).

Secondly, our definition does not exclude the reconfiguration of components with an unknown state which could be problematic in specific cases. Thirdly, usually broken components are not candidates for reconfiguration, unless they can be reset, and one may want to ignore them.

Computation

The analogy between consistency-based reconfiguration and diagnosis expressed by Definition 10.20 suggests how solutions to the characterization and computation of diagnoses may carry over to reconfiguration. If, for a given (set of) mode assignment(s) and observations, a state assignment is inconsistent with *GOALS*, in any reconfiguration at least one of the assigned states has to be modified. For instance, the set of open switches that together isolate a consumer from the generators produce an inconsistency with the goal of supplying this customer, and at least one of them must be closed. Reconfigurations can be calculated from such (minimal) “state conflicts”.

The computation of consistent mode and state assignments could be done jointly. However, the minimality (or preference) of assignments will only apply to modes, since, usually, there is no distinction between states that is analogous to “OK vs. faulty” for modes.

Since the effect of a proposed state change has to be checked for consistency explicitly (changing the position of a switch may connect one consumer, but disconnect another one, which causes an inconsistency regarding another goal), the problem is equivalent to fault identification. Search heuristics and (cost-based) preferences are important, and the problem has triggered the generalization of the algorithm used in *SHERLOCK* (see Section 10.4.1) to “*conflict-directed A** search” [86].

Reconfiguration planning

What we have defined as a reconfiguration, is, stated more precisely, the goal state of the reconfiguration. In most cases, achieving this goal is not a straightforward task, such as simply changing switch positions in an arbitrary order. The individual state changes may require a sequence of low level actions. Often, there are constraints on the order of the reconfiguration actions (e.g., first activate a generator, then change the topology). Also, (temporary) state changes that are not directly implied by the goal reconfiguration may be required. This may result from intermediate goals, safety criteria and restrictions. For instance, reconfiguration of a power network has to avoid states that cause an overload to individual lines. It can be the case that a perfect goal state cannot be achieved by a plan that respects all intermediate restrictions. As a result, planning is needed to turn a computed reconfiguration into a sequence of executable actions [3, 44].

10.6.3 Process-oriented Therapy Proposal

In contrast to component-oriented reconfiguration, which generates remedies exploiting the given system structure, a process-oriented model supports a more general class of therapies, which may include structural modifications of the system (model) [41, 72].

An appropriate treatment of the problem of an increased concentration of iron in drinking water is to add some oxidizer, such as ozone or chlorine, in the plant. This corresponds to an extension of the model: an object (substance) is added, triggering an oxidation process, which in turn produces a (potentially) new structural element, iron oxide, etc. Again, the search aims at a model that is consistent with therapy goals. In contrast to diagnosis (situation assessment), the introducibles for the possible model revisions are not origins of disturbances, but due to human intervention. The library has to contain interventions. They can be modeled as processes with conditions that simply correspond to the decision to perform the respective intervention. These “action triggers” can syntactically be introduced as objects and are the introducibles for the therapy search. The task of finding a therapy for a given situation assessment is then formalized as a search for a (minimal) set of "action triggers" that modify the model such that it becomes consistent with (or entails) the therapy goals, $GOALS_T$. Again, the question arises which part of the information about the current situation persists and which one becomes obsolete due to the intervention. Process-oriented modeling suggests a solution in which the stimulation of additional processes can only cause continuous changes of quantities, i.e., the absolute values of quantities persists (and so do the existing objects), but their derivatives may change. In the water treatment scenario, the oxidizing process does not cause a discontinuous jump of the iron concentration below the threshold, but turns its derivative negative. In fact, this appears to be a natural formulation of therapy goals in this context: if a quantity has an undesired deviation, a goal is forcing its derivative to an opposite sign. For the assumptions $ASSM'$ in a situation assessment, their persistent part $ASSM'_P$ needs to be determined.

Definition 10.21 (Process-oriented therapy). *Let*

$$SIT_P = (STRUCTURE, QUANT_P, ASSM'_P) \cup OBS$$

be the persistent part of a situation assessment and the observations.

A set of action triggers DEC is called a consistency-based therapy for SIT_P and a set of therapy goals $GOALS_T$, if it is consistent with SIT_P and $GOALS_T$:

$$DEC \cup SIT_P \cup LIB \cup GOALS_T \not\models \bot.$$

DEC is called a minimal therapy, if it is minimal with respect to set inclusion among the set of therapies.

Specifying the therapy goals may not be as straightforward as it appears. On the one hand, there are therapy goals related to the violated ones in the current situation ("reduce the concentration of dissolved iron"). On the other hand, a therapy should not sacrifice other goals, which are maintained in the current situation (for instance, achieving a reduction of the iron concentration by stopping the pumps that transport water into the plant is definitely in conflict with the maintenance of a certain amount of supply to the city). Secondly, it may be impossible to achieve all therapy goals in a single step, and, hence, one has to find a therapy that achieves a subset of them, a maximal one, the most critical ones, etc. In this case, a trade-off needs to be found between minimizing *DEC* and optimizing the set of satisfied goals.

Note that decisions need to specify a location for the respective intervention. For instance, one needs to distinguish the (probably preferred) decision of adding an oxidizer in the tank from the decision to do so in the reservoir. This can be achieved by exploiting the spatial relations needed for located objects in general.

Finally, it has to be pointed out that the solution outlined here takes a static perspective on therapy (analogously to situation assessment) and does not address the task of planning a sequence of interventions needed to ultimately achieve a set of goals.

10.7 Other Tasks

10.7.1 Configuration and Design

In the previous sections, we mainly looked at tasks that are concerned with some faulty or unwanted behavior of a system. As we stated before, this reflects a major focus of the field and also the fact that the existing solutions are the most advanced ones. At a first glance, it may sound counterintuitive that handling the many ways in which systems might fail should be easier to solve than, for instance, a design task, in which commonly only the OK behavior is regarded. After all, in Section 10.2, we pointed out the general common denominator of diagnosis and design: searching for a model that is consistent with the observations or the goal specification, respectively. It is useful to analyze the preconditions that make diagnosis manageable, in order to understand what can make design hard in general or feasible in special cases. The main reasons are probably the following: In component-oriented diagnosis

1. the **structure** of the system is usually **fixed**. The search space defined by the considered fault modes of components and finite, although potentially huge. (If the structure is subject to variation, e.g., due to unforeseen component interactions or in process-oriented diagnosis, the task becomes more difficult to solve),

2. there exists a **good initial hypothesis** (namely the OK mode assignment), and the proper diagnosis is only a few revision steps away, due to a plausible minimality criterion,

3. **observations** can effectively **reduce the search space**.

In contrast, design in the most general sense includes **finding** an appropriate structure, which turns the search space infinite in principle. This might be overcome when there exists a good initial design hypothesis not too far from an existing solution. This could even exist in innovative design, for instance, provided by analogy to a solution in some other domain (based on the correspondence of mechanical, electrical, hydraulic laws) [82, 83]. However, most real design tasks in industry are more routine and often provide restrictions that allow for the exploitation of the diagnostic techniques. In many situations, the structure of a solution is given as the one of a similar device or a basic structure plus a limited set of possible modifications (variant design). Or the structure is fixed, and the task is to refine it by specialization of the component types and connections and/or choice of parameters (configuration and parametric design). However, systems supporting such tasks are usually not based on explicit behavior models of the available component types. Instead, the requirements for achieving a certain functionality are directly expressed as interdependencies among component types, parameters of components, restrictions on viable structures, etc. A typical example, the configuration of telephone switching systems, is described in [28]. A configuration *CONF* can be understood as a specification of the structure and parameters of a system,

$$\mathit{CONF} = \mathit{STRUCTURE} \cup \mathit{PARS}$$

(in the same sense as for diagnosis) that respects all general constraints in the respective configuration domain, the domain description *DD*, and is consistent with a specification of the configuration goals, *CONFGOALS* [29].

Definition 10.22 (Configuration). *CONF is a configuration for a domain description, DD, and configuration goals, CONFGOALS, if*

$$DD \cup CONF \cup CONFGOALS \not\models \bot.$$

This suggests the analogy to diagnosis and the general design task, but emphasizes also that *DD* may not fully specify the structure (contrary to *SD* in diagnosis), which becomes part of the solution to be generated.

Diagnostic techniques may be of help to generating designs in identifying the design decisions underlying the inconsistency of some intermediate design result with the *GOALS* (in analogy to conflicts in diagnosis). This may support the human designer in identifying decisions that need to be revised in order to approach a solution.

An example is work on the debugging of hardware designs in [37], which uses a *VHDL* (*Very High Speed Integrated Circuit Hardware Description Language*) to describe the design *GOALS* to be checked against the simulation of a layer in this hierarchical description.

10.7.2 Failure-Modes-and-Effects Analysis

A subtask in the design process that can be supported by model-based systems is failure-model-and-effects analysis (FMEA). The core of this task, which is widespread and standardized to some extent in military, aeronautics, and automotive industries, is to determine the impact each possible component fault may have on the functionality and then to assess the severity and detectability, which, together with the fault probability, determines its criticality and the demand for potential design changes.

Model-based support is feasible, since the design is given and usually, the analysis considers only single faults (or double faults if a single fault can be masked). As another input to the analysis, the user can specify the relevant functions or directly the unwanted violations of the functions, the effects, *EFFECTS*, as well as possibly a set of different scenarios to which the analysis should be applied.

Given a library with fault models, a fault *F* causes the effect, *EFFECT*, in a scenario, *SCEN*, if

$$SD \cup \{MA_F\} \cup SCEN \models EFFECT,$$

and may cause it if

$$SD \cup \{MA_F\} \cup SCEN \cup EFFECT \not\models \bot$$

(see [55]).

Alternatively, the model-based system can compute the behavior for the OK case and derive effects as any deviation of the fault model with respect to some functionally relevant variable [59].

10.7.3 Debugging and Testing of Software

At a first glance, it seems to be straightforward to apply the (component-oriented) diagnosis techniques to a special class of artifacts, namely software, and, thus, provide model-based tools for the debugging of programs. However, a proper analysis of the task reveals that there are substantial differences compared to diagnosis of physical devices. While there are straightforward and justified ways to consider a program to be structured into components (modules, functions, procedures, lines of code, ...), a number of assumptions underlying most consistency-based diagnosis theories and techniques (as discussed in Section 10.4.1) are violated in principle:

- **Component faults only:** A wrong behavior of a program often cannot be blamed to any of the existing "components", but may be caused by some **missing** step or computation.

- **No structural faults:** This is violated because of the fact stated above, but possibly even for a more fundamental bug in the overall structuring of the program.

- **Well-designed system:** This does not hold for principled reasons: after all, debugging of a program becomes necessary **because** it is **not** well-designed!

While the physical device is assumed to have worked properly before some component(s) broke, the program has never performed correctly (and never will).

Stated systematically, software debugging is not a special case of diagnosis, but an instance of the task of identifying flaws in a **design**. This implies, in particular, that the behavior specification contradicts the assumption that all system components are *OK*, instead of being implied by it. As a consequence, the intended behavior, *GOALS*, has to be made explicit and checked against the results produced by the program:

$$\mathit{MODEL} \cup \mathit{GOALS} \models \bot.$$

The usual ways for capturing *GOALS* are by an explicit specification (as an abstract representation of the intended behavior of the program) or, in a more fragmentary way, by a set of tests (which define an input to the system and the expected output or a classification of the actual output as correct or incorrect). Any inconsistencies detected can then be exploited by consistency-based techniques as for fault localization in physical systems, but under caveats that stem from the potentially violated preconditions of these techniques. It is not obvious that fault localization in this style delivers useful results in case structural bugs are present in the program.

Performing fault localization requires an appropriate representation of the structure of the program. While the "components" in this structure could be directly given by the code (lines of code, functions, ...), the interaction among these components (the "connections") have to be generated exploiting the syntax and semantics of the respective programming language. Further structuring (e.g., additional entities in a hierarchy) may be gained from the specification or by abstraction from the code.

Fault models, which enabled fault identification and often tighter fault localization in case of physical devices, are hard to obtain for software and can hardly be expected to be exhaustive, except at a very abstract level of modeling. While in the physical world, the behavior of a faulty (elementary) component is often fairly constrained and predictable, the space of possible bugs in a program is spanned by the creativity of programmers and, hence, practically infinite.

Despite and within these limitations, research on model-based debugging aids has produced encouraging results if only for small programs [43, 50, 51, 85], and also work on fault-model-based software testing is carried out [27].

A related task is debugging of knowledge bases. In [29], consistency-based techniques are applied to localize faults in a knowledge base for a configuration system. Here, the *GOALS* are represented as a set of (positive and negative) examples.

10.8 State and Challenges

The field of model-based systems started off by **building systems** that exploited reasoning from first principles instead of purely experiential knowledge. In contrast to work on diagnosis in engineering, which tends to be very domain or even device-specific, it aimed at generic solutions and focused for a while on developing a rigorous theoretical foundation [62, 19, 69, 24]. Basically, the main part of this work was completed more than ten years ago and is still quite well represented in [39].

While some work at the purely theoretical level has been continued until today, also considerable attempts were made to apply theory and technology to serious real-life and industrial problems during the last ten years or so (see [42, 78, 53, 59, 68, 74, 66, 84]. Most of them resulted in feasibility studies and (often quite advanced) prototypes, but few solutions could be commercialized and used in every-day practice so far. The industrial potential of model-based systems technology has been recognized and is considered plausible. It offers

- a systematic way to generate and adapt solutions based on model libraries;
- model libraries as an important corporate knowledge repository whose elements can be exploited and re-used during the entire life-cycle of a product;
- a reduction of manual work and a guaranteed coverage of a model-based solution;
- the enablement of system autonomy through self-diagnosis and self-reconfiguration.

The existing theory and technology of model-based systems now needs to improve the basis for a transfer into industrial applications. Some of the important issues are

- **distributed/cooperative diagnosis** if local diagnostic capabilities exist and need to be designed, interfaced and exploited (highly relevant, for instance, in the automotive industries);
- diagnosis of **structural faults** if unanticipated interactions occur (e.g., bridge faults in circuits);
- diagnosis of **non-component-oriented systems** (for instance, for applications in process industries or environmental/ecological applications);
- **scaling up** algorithms to handle large systems, for instance, by precompilation steps.

However, the most essential current challenges appear to be the following.

Creation of a theory, methodology, and powerful tools to build libraries of (diagnostic) models

Model-building is a distinctive feature of the technology. All projects that solved a problem relevant to industrial practice had to build component models. However, to our knowledge, none of them developed a serious library of behavior models that could be easily reused in another project. Only few attempts have been made to develop a theory of building diagnostic models (e.g., [69, 67]). Especially, a well-founded theory and methodology for developing **reusable** model libraries is needed. For industrial applications, the creation of such libraries is crucial. If generating a system model cannot be done easily based on a library, the model-based diagnostic algorithms may be rendered useless, because encoding diagnostics by hand may be cheaper.

What is the problem? Usually, the models used for successful projects have some specificities of the diagnostic task, domain, or even device compiled into them. While this is justified and can even be essential for obtaining an efficient solution, it prevents the reuse of the models even in similar applications where some of the modeling assumptions do not hold. Including the most general descriptions, which cover all potentially relevant features of a component's behavior in the library can lead to overloaded models and useless predictions in each single application. For instance, a model of a pipe in the air intake of a vehicle engine may need to include the oxygen concentration, whereas a pipe model in the exhaust system has to propagate emissions. The model of a pipe that supplies a control valve with pressure, however, should do just this, rather than involving the oxygen and CO_2 concentration. Especially the field of qualitative modeling is challenged to produce solutions that are of help for **effectively and efficiently** producing libraries of reusable model fragments and generating tailored system models using their fragments for **industrial practice**. Such solutions also have to include a methodology and tools for the **distributed** production and maintenance of such libraries.

Furthermore, the challenge includes the problem of incorporating numerical distinctions and models. On the one hand, many diagnostic tasks require distinctions between different modes based on numerical thresholds. This enforces numerical distinctions in model fragments that are not determined locally, but by a specific context, the necessary distinctions in other component models, the structure, the precision of observations, to name a few important factors. The second reason is that numerical models of systems and components often do exist in industrial practice that can and must be exploited and placed in a well-defined relation to the more abstract diagnostic models.

Involving numerical models and modeling environments in model-based systems solutions is important because they reflect current engineering practice and education, which leads to the second major prerequisite for a systematic exploitation of model-based technology in industry.

Creation of models of the problem solving tasks as work processes that enable the integration of model-based systems into current practice and tool chains

The tasks addressed by model-based systems are not novel. Diagnosis, FMEA, testing, etc. are existing activities of human experts, mainly engineers, often carried out in teams or collaboration and supported by a variety of (software) tools, such as CAD tools, workshop testers, FMEA-editors. A model-based system is no solution, if it is not a solution to effectively supporting these work processes in real practice. Hardly any product offered by model-based technologies can claim to aim at the complete automation of some task. And even if it does, it depends on appropriate input, a model library being the minimum, and it has to deliver results in a way that supports interaction with its environment, a physical system, human agents, or an organization. Offering real support requires, among other issues,

- that the user concepts and perspectives on the system and the task are properly reflected in the model-based system,
- that the required input to the system can (easily) be made available in practice and the results are of a kind and form that can be further processed by other tools and/or people,
- that the system actually addresses the **difficult or costly** steps in the workflow.

The last issue is crucial to the application of a technology. The model-based problem solvers of today are mostly concerned with the formalization and automation of **reasoning** tasks (such as diagnosis in the above sense, test generation, etc.), but these tasks are sometimes not too difficult or cheap. However, the **actions** involved in finding and removing faults essentially determine the costs (and downtime). An automated system for workshop diagnosis will only pay off, if it helps to generate good plans for the required activities, which reduce the costs.

What is required is a scientific analysis and formalization of the tasks and the concepts and activities of human organizations to master them. The field now needs to move forward from developing abstract problem solving algorithms to developing **models of work processes** the algorithms and problem solvers have to be embedded in and to studying the practical context of developing model-based solutions and, in particular model libraries.

Acknowledgements

I would like to thank for various kinds of support, especially Gautam Biswas, Johan de Kleer, Oskar Dressler, Michael Esser, Alessandro Fraracci, Gerhard Friedrich, Michael Gelfond, Dominik Goby, Lukas Kuhn, Markus Stumptner, Christian Unger, Brian Williams, and Franz Wotawa. Special thanks to the editors of this handbook for their efforts and their patience.

Bibliography

[1] G. Biswas, M.O. Cordier, J. Lunze, M. Staroswiecki, and L. Trave-Massuyes. *IEEE SMC Transactions, Part B*, 34, 2005 (Special Volume on Diagnosis of Complex Systems: Bridging the Methodologies of the FDI and DX Communities).

[2] A. Beschta, O. Dressler, H. Freitag, M. Montag, and P. Struss. A model-based approach to fault localization in power transmission networks. *Intelligent Systems Engineering*, 2, 1993.

[3] M. Balduccini and M. Gelfond. Diagnostic reasoning with a-Prolog. *Theory and Practice of Logic Programming*, 3, 2003.

[4] B. Bredeweg and P. Struss (guest eds.). Qualitative reasoning. *AI Magazine*, 2004.

[5] G. Brewka, I. Niemela, and M. Truszczynski. Nonmonotonic reasoning. In V. Lifschitz, B. Porter, and F. van Harmelen, editors. *Handbook of Knowledge Representation*. Elsevier, 2007.

[6] M. Chantler, S. Daus, T. Vikatos, and G. Coghill. The use of quantitative dynamic models and dependency recording engines. In *7th International Workshop on Principles of Diagnosis (DX-96)*, 1996.

[7] J.W. Collins. Process-based diagnosis: An approach to understanding novel failures. PhD thesis, Institute for the Learning Sciences, Northwestern University, 1993.

[8] J. Crow and J. Rushby. Model-based reconfiguration: Toward an integration with diagnosis. In *Proceedings of AAAI-91*, 1991.

[9] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3), 1991. Also in [39].

[10] A. Darwiche. Compiling devices: A structure-based approach. In *Principles of Knowledge Representation and Reasoning (KR 98)*, 1998.

[11] R. Davis. Expert systems: Where are we? and where do we go from here? *Artificial Intelligence*, 3(2), 1982.

[12] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 1984. Also in [39].

[13] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[14] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28, 1986.

[15] D. Dvorak and B.J. Kuipers. Model-based monitoring of dynamic systems. In *International Joint Conference on Artificial Intelligence*, 1989.

[16] J. de Kleer. Modeling when connections are the problem. In *Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[17] J.D. de Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24, 1984. Also in [81].

[18] J. de Kleer and J. Kurien. Fundamentals of model-based diagnosis. In *Proceedings of SafeProcess03*, 2003.

[19] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56, 1992.

[20] J.D. de Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 31(1), 1987. Also in [39].

[21] J. de Kleer and B.C. Williams. Compiling devices: Locality in a TMS. In B. Faltings and P. Struss, editors. *Recent Advances in Qualitative Physics*. MIT Press, 1992.

[22] J. de Kleer and B.C. Williams. Diagnosis with behavioral modes. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1993. Also in [39].

[23] O. Dressler. On-line diagnosis and monitoring of dynamic systems based on qualitative models and dependency-based diagnostic engines. In *Proceedings of the European Conference on Artificial Intelligence*, 1996.

[24] O. Dressler and P. Struss. Back to defaults: Characterizing and computing diagnoses as coherent assumption sets. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-92)*, 1992.

[25] O. Dressler and P. Struss. Model-based diagnosis with the default-based diagnostic engine: Effective control strategies that work in practise. In *11th European Conference on Artificial Intelligence*, 1994.

[26] O. Dressler and P. Struss. A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics. In *Proceedings of the 14th International Workshop on Principles of Diagnosis*, June 2003.

[27] M. Esser and P. Struss. Fault-model-based test generation for embedded software. In *International Joint Conference on Artificial Intelligence*, 2007.

[28] G. Fleischanderl, G. Friedrich, A. Haselboeck, H. Schreiner, and M. Stumptner. Configuring large systems using generative constraint satisfaction. *Intelligent Systems Archive*, 13(4), 1998.

[29] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152, 2004.

[30] G. Friedrich, G. Gottlob, and W. Nejdl. Formalizing the repair process. In *Proceedings of the 10th European conference on Artificial intelligence*, 1992.

[31] G. Friedrich, G. Gottlob, and W. Nejdl. Physical impossibility instead of fault models. In [39], 1992.

[32] M. Fisher. Temporal representation and reasoning. In V. Lifschitz, B. Porter, and F. van Harmelen, editors. *Handbook of Knowledge Representation*. Elsevier, 2007.

[33] G. Friedrich and F. Lackinger. Diagnosing temporal misbehavior. In *IJCAI-91*, 1991.

[34] K. Forbus. Qualitative process theory. *Artificial Intelligence*, 24, 1984. Also in [81].

[35] K. Forbus. Qualitative reasoning. In *Handbook of Knowledge Representation*. Elsevier, 2008.

[36] B. Faltings and P. Struss. *Recent Advances in Qualitative Physics*. MIT Press, 1992.

[37] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 3, 1999.

[38] R. Greiner, B.A. Smith, and R.W. Wilkerson. A correction to the algorithm in Reiters theory of diagnosis. *Artificial Intelligence*, 41, 1989.

[39] W. Hamscher, J. de Kleer, and L. Console, editors. *Readings in Model-based Diagnosis: Diagnosis of Designed Artifacts Based on Descriptions of their Structure and Function*. Morgan Kaufmann, 1992.

[40] U. Heller. Process-oriented consistency-based diagnosis-theory, implementation and applications. PhD thesis, Akademische V.-G., 2001.

[41] U. Heller and P. Struss. Consistency-based problem solving for environmental decision support. *Computer-Aided Civil and Infrastructure Engineering*, 17, 2002.

[42] G. Karsai, G. Biswas, S. Narasimhan, T. Szemethy, G. Peceli, G. Simon, and T. Kovacshazy. Towards fault-adaptive control of complex dynamic systems. In T. Samad and G. Balas, editors. *Software-Enabled Control: Information Technologies for Dynamical Systems*. Wiley–IEEE Press, 2003.

[43] D. Koeb and F. Wotawa. Fundamentals of debugging using a resolution calculus. In *International Conference on Fundamental Approaches to Software Engineering (FASE)*, *LNCS*, vol. 3922. Springer, 2006.

[44] P. Kim, B. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[45] F. Lin. Situation calculus. In V. Lifschitz, B. Porter, and F. van Harmelen, editors. *Handbook of Knowledge Representation*. Elsevier, 2007.

[46] G. Lamperti and M. Zanella. *Diagnosis of Active Systems—Principles and Techniques*. Kluwer Academic Publisher, 2003.

[47] P. Mosterman and G. Biswas. A comprehensive methodology for building hybrid models of physical systems. *Artificial Intelligence*, 121, 2000.

[48] I. Mozetic. Hierarchical model-based diagnosis. *Int. J. of Man-Machine Studies*, 35, 1991.

[49] S. McIlraith and R. Reiter. On tests for hypothetical reasoning. In [39], 1992.

[50] W. Mayer and M. Stumptner. Extending diagnosis to debug programs with exception. In *IEEE Automated Software Engineering Conference*, 2003.

[51] W. Mayer and M. Stumptner. Abstract interpretation of programs for model-based debugging. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[52] P. Nayak. *Automated Modeling of Physical Systems*. Springer, 1995.

[53] S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *IEEE Trans. on Systems, Man, and Cybernetics, Part A*, 37(3), 2007.

[54] Y. Pencole and M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164, 2005.

[55] C. Picardi, L. Console, F. Berger, J. Breeman, T. Kanakis, J. Moelands, S. Collas, E. Arbaretier, N. De Domenico, E. Girardelli, O. Dressler, P. Struss, and B. Zilbermann. AUTAS: a tool for supporting FMECA generation in aeronautic systems. In *Proceeding of the 16th European Conference on Artificial Intelligence*, 2004.

[56] B. Pulido and C.A. Gonzalez. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(5), 2004.

[57] D. Poole. Normality and faults in logic-based diagnosis. In *11th International Joint Conference on Artificial Intelligence*, 1989. Also in [39].

[58] G. Provan and D. Pool. The utility of consistency-based diagnostic techniques. In *Proc. Second International Conference on Principles of Knowledge Representation and Reasoning*, 1991.

[59] C. Price. Autosteve: Automated electrical design analysis. In *Proceedings ECAI-2000*, 2000.

[60] O. Raiman, J. de Kleer, V. Saraswat, and M. Shirley. Characterizing non-intermittent faults. In *Proceedings of AAAI-91*, 1991. Also in [39].

[61] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13, 1980.

[62] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1), 1987. Also in [39].

[63] F. Rossi, P. van Beek, and T. Walsh. Constraint programming. In *Handbook of Knowledge Representation*. Elsevier, 2008.

[64] P. Struss and O. Dressler. Physical negation—integrating fault models into the general diagnostic engine. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.

[65] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event system. *IEEE Transactions on Automatic Control*, 43(7), 1998.

[66] P. Struss and C. Price. Model-based systems in the automotive industry. *AI Magazine*, 24, 2004.

[67] M. Sachenbacher and P. Struss. Task-dependent qualitative domain abstraction. *Artificial Intelligence*, 162, 2005.

[68] M. Sachenbacher, P. Struss, and R. Weber. Advances in design and implementation of OBD functions for diesel injection systems based on a qualitative approach to diagnosis. In *SAE 2000 World Congress*, Detroit, USA, 2000.

[69] P. Struss. What's in SD? Towards a theory of modeling for diagnosis. In [39], 1992.

[70] P. Struss. Testing for discrimination of diagnoses. In *5th International Workshop on Principles of Diagnosis*, 1994.

[71] P. Struss. Fundamentals of model-based diagnosis of dynamic systems. In *15th International Joint Conference on Artificial Intelligence*, 1997.

[72] P. Struss. Artificial intelligence methods for environmental decision support. In *e-Environment: Progress and Challenge*, 2004.

[73] P. Struss. Automated test reduction. In *19th Int. Workshop on Qualitative Reasoning. 16th International Workshop on Principles of Diagnosis*, 2005.

[74] P. Struss. A model-based methodology for the integration of diagnosis and fault analysis during the entire life cycle. In *Proceedings Volume from the 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess06)*, 2006.

[75] Y. Sun and D. Weld. A framework for model-based repair. In *Proceedings of AAAI-93*, 1993.

[76] M. Tiller. Introduction to Physical Modeling with MODELICA. *The Springer International Series in Engineering and Computer Science*, vol. 615. Springer, 2001.

[77] L. Trave-Massuyes, M.O. Cordier, and X. Pucel. Comparing diagnosability in CS and DES. In *Proceedings Volume from the 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess06)*, 2006.

[78] L. Trave-Massuyes and R. Milne. TIGER$^{\text{TM}}$—gas turbine condition monitoring using qualitative model-based diagnosis. *IEEE Expert*, 12(3), 1997.

[79] Y. Umeda, T. Tomiyama, H. Yoshikawa, and Y. Shimomura. A design methodology for self-maintenance machines. *IEEE Expert: Intelligent Systems and their Applications Archive*, 9(3), 1994.

[80] I. Vatcheva, O. Bernhard, H. de Jong, and N.J.I. Mars. Experiment selection for the discrimination of semi-quantitative models of dynamical systems. *Artificial Intelligence*, 170, 2006.

[81] D. Weld and J. de Kleer. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, 1990.

[82] B. Williams. Interaction-based invention: Designing novel devices from first principles. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[83] B. Williams. Interaction-based invention: Designing novel devices from first principles. In [36], 1992.

[84] B. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, 1996.
[85] F. Wotawa. On the relationship between model-based debugging and program slicing. *Artificial Intelligence*, 135, 2002.
[86] B. Williams and R. Ragno. Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Math.*, 2003.

This page intentionally left blank