Institut für Informatik der
Technischen Universität München
Lehrstuhl Informatik IX

Automated Qualitative Abstraction and its Application to Automotive Systems

Martin Sachenbacher

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Dr.h.c. Wilfried Brauer
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Peter Struss
	2. O.Univ.-Prof. Dr. Georg Gottlob, Technische Universität Wien (Österreich)

Die Dissertation wurde am 12. 4. 2001 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 5. 7. 2001 angenommen.

Abstract

The increasing complexity of engineered devices, e.g. in the domain of automotive systems, has lead to an increased demand for computer-supported behavior prediction, diagnosis, and testing. Model-based reasoning is an emerging subfield of Artificial Intelligence that is concerned with representing knowledge about the structure and behavior of physical systems in terms of a model and using it to automate the above-mentioned tasks.

Modeling is the hard part of model-based reasoning. In order to make it feasible, it is crucial to break down the knowledge about a device into re-usable components and to organize them in a library. On the other hand, in order to keep reasoning with a model computationally tractable, it is important to have an adequate representation that includes only the distinctions that are required to perform a particular task.

In this thesis, we deal with the problem of finding a level of granularity for a behavior model that is as coarse as possible, but still fine enough for a given behavior prediction or diagnosis task. The focus is on task-dependent domain abstraction: i.e., the problem is to determine distinctions within the domains of variables (termed qualitative values) that are both necessary and sufficient, given the constraints of the behavior model, a granularity of possible observations, and a granularity of desired results. We present a method that allows to automatically determine qualitative values, starting from a base model that has been composed from a library.

A principled application of this work is to turn real-valued models, as commonly used in industry, into qualitative models to make them accessible to automated reasoning methods. The resulting methods and software tools thus greatly enhance the ability to use a behavior model of an engineered device as a common basis to support different tasks along its life cycle. The thesis describes the application to real-world examples taken from the automotive domain. This leads to the first model-based diagnosis system running on-board a passenger vehicle. The prototype is shown to provide useful results for a number of emission-related failure scenarios that were implemented on a Volvo demonstrator car.

Zusammenfassung

Die zunehmende Komplexität technischer Systeme, z.B. im Automobilbereich, führt zu einem steigenden Bedarf an computerunterstützter Verhaltensvorhersage, Diagnose und Testgenerierung. Modellbasiertes Schließen ist ein Teilgebiet der Künstlichen Intelligenz, das sich mit der Repräsentation von Wissen über Struktur und Verhalten physikalischer Systeme in Form eines Modells und dessen Verwendung zur Automatisierung der genannten Aufgaben beschäftigt.

Der Modellierungsschritt stellt die Hauptschwierigkeit bei modellbasiertem Schließen dar. Er ist nur dann effizient durchführbar, wenn das Wissen über ein Gerät durch möglichst allgemeine, wiederverwendbare Komponentenmodelle repräsentiert wird, die in Form einer Modellbibliothek organisiert und so für verschiedene Aufgaben herangezogen werden können. Andererseits erfordert effizientes Problemlösen mit einem Modell eine angemessene Repräsentationsebene, in der nur Unterscheidungen berücksichtigt werden, die zur Durchführung einer gegebenen Aufgabe tatsächlich notwendig sind.

Diese Arbeit beschäftigt sich mit dem Problem, eine Granularitätsebene für ein Verhaltensmodell zu finden, die so grob wie möglich, aber noch fein genug für eine gegebene Verhaltensvorhersage- oder Diagnoseaufgabe ist. Den Schwerpunkt bildet dabei die aufgabenabhängige Wertebereichsabstraktion: d.h. das Problem besteht darin, Unterscheidungen innerhalb des Wertebereichs von Variablen (sogenannte qualitative Werte) zu finden, die sowohl notwendig als auch hinreichend sind, bezogen auf die Constraints des Verhaltensmodells, die Granularität der möglichen Beobachtungen, und die Granularität der gewünschten Ergebnisse. Wir stellen eine Methode vor, mit der qualitative Werte automatisch bestimmt werden können, ausgehend von einem Basismodell, das aus einer Modellbibliothek erzeugt worden ist.

Eine prinzipielle Anwendung besteht darin, numerische Modelle, wie sie häufig in der Industrie verwendet werden, in qualitative Modelle zu transformieren und dadurch den Methoden des automatischen Schlußfolgerns zugänglich zu machen. Die resultierenden Verfahren und Software-Werkzeuge ermöglichen es, ein Verhaltensmodell als gemeinsame Basis zur Unterstützung verschiedener, während des Lebenszyklus eines technischen Systems anfallender Aufgaben einzusetzen. Die Arbeit beschreibt dies anhand von realen Beispielen aus dem Automobilbereich. Dies führt schließlich zum ersten modellbasierten Diagnosesystem, das on-board auf einem Personenfahrzeug läuft. Es wird gezeigt, daß dieser Prototyp nützliche Ergebnisse für eine Reihe von emissionsrelevanten Fehlerszenarien liefert, die auf einem Volvo-Versuchsfahrzeug implementiert wurden.

Acknowledgments

Many people have contributed to this work. This supports my general belief that scientific work often arises from the interaction with a creative "environment", rather than from the mind of a single person.

For the scientific part of this environment, I first would like to thank my supervisor, Peter Struss. He has a particular knack for structuring and organizing research on a conceptual and general level, while never losing sight of its practical applications. Many aspects of this work were influenced by his ideas, while at the same time its presentation was clarified through his suggestions. He gave me the freedom to pursue my interests, and provided encouragement and guidance throughout my work on this thesis.

I would also like to thank my former colleagues at the Technische Universität München, Ulrich Heller, Andreas Malik and Jakob Mauss, for interesting discussions, collaboration, and in general for sharing the fun of graduate student life with me. Oskar Dressler from Occ'm Software helped to improve this work by many discussions and provided continuous support and hints for integration with the Raz'r model-based reasoning framework, fairly going beyond the scope of normal "customer care". Alexander Kutscha implemented core parts of the software presented in this thesis, its GUIs, and its interfaces to Raz'r. Part of the models used in the examples are based on a model library developed by Florian Dummert during his master thesis. Andreas Malik implemented the parts of the signal transformation module that manage the interfacing with vehicle data. Finally, I would like to thank the scientific community around the DX and QR workshops at which I was able to participate. My thesis benefitted from comments of reviewers on papers that presented aspects of this research, as well as from discussions arising during these events. Of the many people, I would like to mention Daniele Theseider–Dupré for his valuable feedback.

A main feature of this work is that it ranges from basic theoretic ideas to a real-world application. This is not an easy task and requires a "milieu" also on part of the application domain; without this, the evaluation of the research ideas would have been impossible. This environment consisted of the German project INDIA (Intelligent Diagnosis in Industrial Applications) and the European project VMBD (Vehicle Model-based Diagnosis, see figure 1). During the latter project, I was a scholarship holder of the Robert Bosch GmbH at Stuttgart, Germany. I am indebted to Wolfgang Bremer, Michael Walther, and Reinhard Weber from Bosch's research and development department. Volvo Car Corporation (now part of Ford Motor Company) at Göteborg, Sweden contributed to analyzing the requirements of the guiding application, selecting the demonstrator scenarios, and preparing the demonstrator car. During several visits, Claes Carlén and Björn Svensson helped me with installations in the car, took measurements, and even allowed me to test drive the car in the landscapes surrounding Göteborg. They and their families made my visits to Göteborg a particular enjoyable experience. I appreciate this a lot as I know now that such people are usually very busy individuals.

Last but not least, I would like to thank Franz Wotawa, who commented on earlier drafts of this manuscript. Finally, another general belief of me is that while the third factor — the private milieu of friends and family — is equally important, it should be omitted from acknowledgments.

Figure 1: Participants and the two demonstrator cars of the VMBD project (image taken by Claes Carlén)

Martin Sachenbacher
January 2001

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) is a science concerned with problem solving by the means of computers. Model-based reasoning is a subfield of AI concerned with representing knowledge about the structure and behavior of physical systems, and using it to automate engineering tasks such as behavior prediction, diagnosis, and testing. It is based on the idea that once the essential aspects of a system have been captured in the form of a behavior model, the tasks mentioned above can be performed using well-understood and computer-supported methods.

However, performing effective problem solving requires, in the first place, a problem representation that is adequate for the task at hand. For instance, when diagnosing the behavior of an engineered device in an on-board environment, it is crucial to have a model that focuses on the essential aspects of the system only, in order to meet the stringent time and space requirements of this application. To put it short, *modeling* is the hard part of model-based reasoning.

Despite its importance, the problem of automatically deriving representations that are suited for a certain task still remains, for a large part, an unsolved problem. As David Waltz put it during his presidential address at the Fifteenth National Conference on Artificial Intelligence ([Wal99]):

> "AI has for the most part focused on logic and reasoning in artificial situations where only relevant variables and operators are specified and has paid insufficient attention to processes of reducing the richness and disorganization of the real world to a form where logical reasoning can been applied. (...) AI models already assume that somehow we've discovered the objects and relations that are important in a situation. Very little work has been done on the problem of actually turning real, complex scenes into symbolic situation descriptions. And I would argue that this is where most of intelligence really lies."

An important idea in order to make modeling of physical systems feasible is to break down the knowledge about a device into re-usable components, and to organize them in a library. A model of the system can then be assembled by composing model fragments from the library, based on a structural description

of the system. This requires that the model fragments have to be formulated, as far as possible, in a generic way and independent of their application context.

These objectives immediately create a tension, because what is an adequate model depends on the structure of the system and the specific task it will be used for — a kind of information that cannot be anticipated in generic model fragments. Hence, in general, the composition of models from a library will lead to system models that are either inefficient, because they are overly detailed (i.e. too fine-grained), or ineffective, because they are not detailed enough for the task at hand (i.e. too coarse-grained). What we would like to have instead is a model that has just the *right* granularity in the sense that it makes all the sufficient, but only the necessary distinctions to solve a particular task.

The problem of task-dependent modeling can only be investigated in the context of concrete application tasks and concrete physical systems. This seems obvious — but, as one of the key persons in the field of model-based reasoning, Johan de Kleer, has noted ([dK93]):

> "Much AI research falls into the trap of examining issues for their own sake, losing sight of the overall objective, and thereby effectively building bridges over dry land . (...) Focusing on reasoning about the physical world constantly brings fundamental issues to attention. Significant AI progress can be made only by applying AI ideas to tasks. Otherwise, we tend to spin our wheels."

This has motivated our basic approach to carry out the work in close relationship to applications, to include industrial partners and experts, and to tackle a real-world example taken from the automotive domain. Because one cannot learn about the problems to be addressed in industrial applications without understanding their nature to some extent, it is also the reason why significant parts of this thesis deal with the function and behavior of vehicle subsystems and issues that are relevant in this particular context.

On the other hand, while doing so, one is faced with the risk of creating specific solutions that have only a limited scope of interest. In order to ensure that the resulting methods are transferable and of more general importance, care has thus to be taken for never leaving the firm ground of mathematical argumentation. This is the reason why considerable sections of this work are concerned with mathematical formalisms, e.g. with the definition of spaces and mappings between elements of these spaces, and mathematical proofs in order to warrant important properties.

In this thesis, we investigate the problem of finding a level of granularity for a behavior model that is as coarse as possible, but still fine enough for a given behavior prediction or diagnosis task. The focus is on task-dependent domain abstraction. I.e., the problem is to determine distinctions within the domains of variables (termed *qualitative values*) that are both necessary and sufficient, given the constraints of the behavior model, the granularity of possible inputs, and the granularity of desired outcomes of model-based problem solving.

In order to automate this aspect of problem solving, it is necessary to lay bare the underlying intuitions and diverse facets of *task-dependency*, and to make them explicit and formal to such a degree that terms like "sufficient" or "necessary" distinctions can be reasoned about.

Based on this, we present a method that allows to automatically determine qualitative values, starting from a base model that has been composed from a library. The resulting methods and software tools enhance the ability to use a behavior model as a common basis to support different tasks. The thesis describes their application to real-world examples taken from the automotive domain. In particular, this leads to the first model-based diagnosis system running on-board a passenger vehicle. This prototype is shown to provide useful results for a number of emission-related failure scenarios that were implemented on a Volvo demonstrator car.

1.1 Motivation

1.1.1 Example

Consider, for example, the system depicted in the figure 1.1 below. The device is a simplified version of a pedal position sensor used in a passenger car. Its purpose is to deliver information about the position of the accelerator pedal to the electronic control unit (ECU) of the engine management system. The ECU uses this information to calculate the amount of fuel that will be delivered to the car engine.

The pedal position is sensed in two ways, via the potentiometer as an analogue signal, v_{pot}, and via the idle switch as a binary signal, v_{switch}. The idle switch changes its state at a particular value of the mechanically transferred pedal position. The two possible values of v_{switch} correspond to two ranges of v_{pot}, separated by a particular voltage value. The reason for the redundant sensing of the pedal position is that the signals from the potentiometer and the switch are cross-checked against each other by the on-board control software of the ECU. This plausibility check is a safety feature of the system, in order to avoid cases where a wrong amount of fuel injected evokes dangerous driving situations.

Assume we want to perform the plausibility check between the electrical signals v_{pot} and v_{switch} automatically by the means of a behavior model of the system, e.g. as part of a diagnosis or design verification task. For many problems, it is convenient to use behavior models of electrical components that distinguish only between the three qualitative values for voltage, e.g.:

$$voltage = \begin{cases} ground & : & voltage = 0V \\ between & : & 0V < voltage < 10V) \\ battery & : & voltage = 10V \end{cases}$$

If we used such a representation for our device, it turns out to be sufficient to reason about simple failures in the electrical harness, for instance, shorts to ground or battery. However, it would be of limited use for the kind of task that

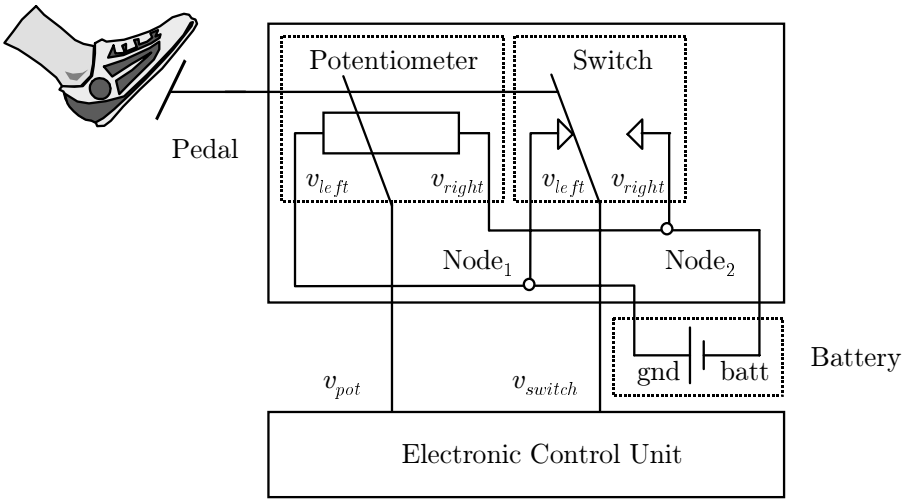

Figure 1.1: The Pedal Position Sensor

we want to perform for the pedal position sensor. The reason is that this task refers to the redundancy which purposefully has been implemented in the system.

For example, consider the case where v_{pot} is above the switching point due to an increased resistance of the potentiometer, but still lower than the battery voltage ($10V$). This failure would remain undiscovered using the representation above. Both the original and the modified value are subsumed by *between*, which is consistent with the switch being either on its left or right position. Thus, this situation cannot be distinguished from the normal operation of the device. Detecting the failure requires an additional distinction in the domain of v_{pot}, which further refines the value *between*. However, this distinction is missing in the domain of voltage, hence the compositional model will be of limited utility for the above-mentioned task.

One could argue that the domain of v_{pot} can easily be replaced by a four-valued domain in order to achieve this additional distinction. The deeper problem, however, is that the particular distinction in the domain of v_{pot} cannot be anticipated in a generic model of the potentiometer, because the voltage landmark would not make any sense in a different structure. It is only the specific combination of the potentiometer and the switch that requires this distinction. Of course, we could use also numerical models that would be able to relate the switching position to a particular division of the potentiometer voltage v_{pot}. However, this model would be overly detailed for the purpose discussed. It would likely be too inefficient to meet the tight space and run-time requirements of being part of the control unit's on-board software, for instance. We would rather like to have a composition of component models that make just the right distinctions as required by the other components of the device and the task the model is intended for.

We might have some intuitions of how this can be achieved. Starting from a basic understanding of how the components in the device behave, and the requirements to achieve a distinction between certain behavioral features, a human problem solver might be able to determine which distinctions he has to make for certain sets of variable values in the system, considering the structure of the device. For instance, the required distinction between *ground* and *batt* for v_{switch} requires the distinction between the two switch states, which in turn determines a distinctive position of the pedal. This, in turn, induces the desired distinction in the domain of v_{pot}.

1.1.2 The Problem

The tiny example above has confronted us with the problem that simply picking model fragments from a library and composing the model is not enough. It is infeasible, in general, to anticipate the required granularity in the model fragments, in particular distinctions in the domains of variables. Instead, the ability to *transform* the model to the right level of abstraction *after* composing them is a crucial requirement.

The problem is important, because it impairs the idea of using a model as a common basis for different tasks. E.g. for automotive systems, it is typical that several tasks along a product's life cycle — such as failure modes and effects analysis, on-board diagnostics development, generation of repair manuals or workshop diagnosis — each share a significant amount of common knowledge about the behavior of the system under consideration. Hence, it would be unacceptable to manually create models from scratch that are tailored to each of these tasks. Instead, one would like to use a common base model and transform it in task-dependent ways such that the required set of tailor-made models is obtained.

This thesis is about automating the transformation of models to a level of abstraction that is adequate for a specific structure and task, much like an engineer's ability to come up with a suitable representation when faced with a certain problem. To do this, we have to answer the following questions:

(1) What is a *behavior model*? How can it be expressed in a formal way? (i.e. what is the representation?)

(2) What is a *model abstraction*? What is the space of possible model abstractions? (i.e. what is the search space?)

(3) How can a *task* be characterized? When is a model abstraction "adequate" for a certain task? (i.e. what is the goal criterion?)

(4) How can we efficiently *construct* task-dependent model abstractions? (i.e. what is the computational search strategy?)

We propose answers to each of the above questions, in particular for domain abstractions and for the two fundamental problem-solving tasks of behavior prediction and diagnosis.

We will turn the developed concepts into an implementation of a system that is able to automatically derive models of physical systems at the required abstraction level and to use them for various application tasks. This provides an answer to the tension between the generality of model libraries (i.e., the requirement of re-usability) vs. the effectiveness of models (i.e., the requirement of task-orientation).

1.2 Objective

From an *application* point of view, it is desirable to integrate and reconcile different tasks that share the same type of knowledge. As outlined above, this can be achieved by defining a base model that captures this common knowledge, and re-using it along the different tasks. Currently, however, methods and computer-supported tools are lacking that would support such an integrated perception of work processes along the various tasks. The methods and tools developed in this thesis are intended as a contribution to enhance this *horizontal integration*, and further alleviate the disruption between different stages of a product's life cycle.

From a *scientific* point of view, the problem of capturing only the "significant aspects" in a model is the central subject of study in the field of qualitative reasoning. Qualitative reasoning is a branch of Artificial Intelligence that combines the quest for computational theories underlying the core skills of engineers with techniques that intend to capture only the essential distinctions in a model of a physical system. Yet, there is still a considerable lack of formal theories that would characterize what the significant distinctions or aspects of models are, and methodologies of how they can be derived. This thesis intends to provide a contribution to this research issue through providing a rigorous definition of *task-dependent qualitative distinctions*, and developing a formal methodology for task-dependent qualitative modeling.

1.3 Overview of Proposed Solution

In this section, we give a brief overview of our answers to the questions raised and the goals outlined in the previous sections. The rest of the thesis develops these ideas in detail.

1.3.1 Model Representation

We will represent models of the behavior of physical systems in a fairly general way. In particular, we do not only treat models composed of continuous or monotonic functions, but also models that contain discrete states and operating modes, as motivated by the example of the switch component in the previous section. This is achieved by representing behavior models as *relations* (constraints) on the parameters and variables $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ — involving input and output variables, internal and state variables — of a system. Accordingly, a model of a component, C, to be analyzed can be stated as a relation

$$RC \subseteq DOM(v_{i_1}) \times DOM(v_{i_2}) \times \ldots \times DOM(v_{i_k}).$$

In this notation, $DOM(v_i)$ denotes the domain of a variable v_i. The domain could consist of infinitely many elements, e.g., the real numbers or intervals reflecting a certain precision, but also finite numbers of elements like the states of a switch, certain behavior or failure modes, or qualitative values. A system description then consists of variables $\boldsymbol{v}$, domains $DOM(v_i)$, and the relations describing the behavior of the individual components. It defines a relation $R(\boldsymbol{v})$ capturing the possible behaviors of the physical system.

This uniform representation allows to cover a broad range of formal or semi-formal behavioral knowledge commonly used in the automotive industry, for instance, equational models or characteristic maps based on measurements of components with complex behavior like the engine. It also provides the basis for a computational solution, which is based on methods that efficiently represent and manipulate finite relations.

1.3.2 Model Abstractions

As noted above, we cannot expect to find models of a component in the library that have the required granularity right away. Rather, the models have to be generated starting from a base model and using some kind of transformation operation that yields as output a more abstract model.

Abstraction might affect each of the constituents of a system description. As demonstrated by the example, a useful type of abstraction can be characterized by dropping "unnecessary" distinctions in a base domain $DOM(v_i)$, such that the abstracted model is formulated in terms of coarser values that possibly contain fewer distinctions than the original base domain. Such transformation operations can be expressed by *domain mappings* that map elements of base domains to elements of transformed domains DOM':

$$\tau_i : DOM(v_i) \to DOM'(v_i) \subseteq 2^{DOM(v_i)}$$

A domain mapping τ_i induces an abstraction of the system model. The idea is that the elements of the domain $DOM'(v_i)$ define qualitative values, i.e. exactly the values that have to be distinguished from each other. The composed base model of the system, together with the domain abstractions for each of the system variables, thus defines the space of possible model abstractions.

1.3.3 Characterization of Tasks

We will be concerned with two fundamental problem-solving tasks on the basis of relational behavior models. *Prediction* is the task of determining restrictions of model variables based on observations (e.g., measurements). *Diagnosis* is the task of revising, based on observations that yield conflicting restrictions for variables, the model in such a way that it becomes consistent again with the observations.

The characteristics of a problem-solving task, i.e. the goal criterion for model abstraction, will be characterized by two different aspects termed *observable distinctions* and *target distinctions*.

Observable distinctions capture what *can* be distinguished in a model, due to the observations that are available — for instance, the granularity of the measurements. Target distinctions capture what *needs* to be distinguished in a model, due to the purpose one is after — for instance, to discriminate faulty behavior modes from correct behavior modes in diagnosis, or to distinguish acceptable deviations from unacceptable deviations of output variables.

In our framework, target distinctions and observable distinctions will both be characterized by abstractions for the domains of variables in the model.

For the pedal position sensor example, the target distinction would separate the ground and battery voltage from the rest of the voltage values for the variable v_{switch} (assuming that the plausibility check of the control unit itself is not represented in the model), while the given observability can be expressed by observable distinctions for v_{switch} and v_{pot}.

1.3.4 Computational Solution

Based on this formalization of the problem, we will be able to characterize solutions, and we show that the general problem of determining qualitative domain abstractions starting from a base model is intractable (i.e., NP-hard). This motivates the development of both an incomplete and a complete solution to the problem. The incomplete solution has the property that the resulting model will, in general, be too abstract: this means it might not contain all of the required distinctions, but the distinctions included in the resulting model are necessary ones.

The algorithmic realization is based on a data structure called *SD Tree* that provides an implicit, hierarchical representation of a model relation in order to avoid combinatorial explosion. The computational complexity of operations on the SD Tree, and thus the efficiency of deriving qualitative domain abstractions, can be bound to structural properties of the system model.

The devised methods have been implemented in the form of several prototypic software modules, which are equipped with graphical user interfaces. They have been integrated into an existing commercial model-based framework for composing models and applying them to the tasks of behavior prediction and diagnosis.

Because we focus on the application to automotive systems, the software module that deals with qualitative abstraction of observations features an interface to a domain-specific measurement acquisition tool that is of wide-spread use in the automotive industry. This enables the application of the tool kit also by people which are not familiar with its underlying theoretical background.

1.4 Contribution

We sum up the main contributions of this work. From a scientific point of view, this thesis makes the following important contributions:

- it introduces a formalization of different tasks, model abstractions, and appropriate models within a common relational framework, making the problem amenable to theoretical analysis.
- it derives a first-principles solution for task-dependent automated qualitative model abstraction and analyzes some of its interesting properties.
- it develops algorithms and data structures as a basis for efficient computation of solutions to the problem on a computer.
- it characterizes principled applications, e.g. deriving significant deviations for model-based prediction or distinctions for model-based diagnosis.

From an application point of view, the prominent research contributions are:

- it develops methods that automatically transform a behavior model to the granularity adequate for a task at hand, thus supporting the re-use of knowledge along various tasks.
- it describes an implementation of the methods in the form of software components, which interact with a commercial model-based prediction and diagnosis framework.
- it combines elements of these components to form the first prototype of a model-based diagnosis system running on-board a passenger vehicle, providing useful diagnostic results for a set of failure scenarios.
- it evaluates these components and prototypes on a couple of examples from the automotive domain, and, in particular, on a real demonstrator vehicle with built-in faults.

1.5 Reader's Guide

This thesis consists of an analysis of the requirements of the application domain (first part), a description of the employed methods and solutions (second part), and empirical results using a computer implementation (third part):

- Chapter 2 starts with a description of automotive systems, the types of systems primarily considered in the thesis, and identifies the main tasks and requirements that arise in this application domain.

- Chapter 3 provides necessary background and describes the state of the art in modeling the behavior of physical systems. It provides a formalization in terms of relational behavior models, which is the foundation to automate the tasks stated in the precedent chapter.
- Chapter 4 introduces the concepts of qualitative descriptions, and discusses the contribution of a various number of existing approaches to the core problem of arriving at a model with adequate (i.e. qualitative) granularity.
- Chapter 5 introduces our framework for task-dependent qualitative model abstraction and analyzes some properties of the solution, in particular, its worst-case computational complexity.
- Chapter 6 is concerned with putting the solution to work. This is done by devising methods that allow to exploit the specific structure of a behavior model in order to make its abstraction computationally feasible.
- Chapter 7 gives an overview of the implemented prototype for task-dependent qualitative model abstraction, and describes principled applications to various types of modeling problems.
- Chapter 8 describes the integration of the techniques into a prototype for model-based on-board diagnosis of an automotive system, and evaluates it on a real demonstrator vehicle.
- Chapter 9 summarizes the results and discusses the achievements and contributions in relation to existing work. Based on this, it identifies directions for future work both for application prototypes and scientific research.

Basic theoretic achievements of this thesis (chapters 5 and 6) have been published in [SS99], [SS00] and [SS01], while results and perspectives for the considered application (chapters 2 and 8) can be found in [SSC00b], [SSW00] and [SSC00a].

Chapter 2

Automotive Systems

In this chapter, we describe automotive systems, which are the primary focus of application in this thesis. Starting from a description of the components and structure of automotive systems, we identify some major tasks and requirements that arise in this application domain. As it will be shown, car industries provides a good example for industrial needs in connection with behavior analysis and diagnosis. This leads us to a characterization of the core problem of qualitative modeling from a work process point of view.

The purpose of this chapter is thus to ground the subsequent work in two respects. First, it is shown that a real application problem of economic importance is tackled in this thesis. Second, the fundamental reasons are revealed for the needs and challenges that will be addressed in the following chapters.

However, the scope of the methods that will be developed is not limited to automotive systems. Automotive systems have been chosen as a typical representative of a class of applications that is characterized by involving large and complex engineered devices. Hence, similar patterns of tasks and requirements might be found e.g. in the domain of aircraft and spacecraft industry, transportation systems, power plants, etc. Readers who are interested in the scientific results only can thus skip this description of the application background and proceed with the formalization developed in the next chapter.

2.1 Types of Components

In a car, all functions, such as accelerating or braking, are achieved by one or more automotive subsystems, such as the drive-train or the braking system ([Bau96]). Whereas these subsystems used to be purely mechanical devices in the past, they are now often combinations of several different physical domains. In order to achieve more functions and to make solutions more cost effective, electronically controlled systems are used in modern cars, consisting of one or more microcomputers that run control software. In the context of automotive systems, these microcomputers are termed *electronic control units* (ECUs). An electronic control unit reads in signals from various sensors and governs actuators in order to

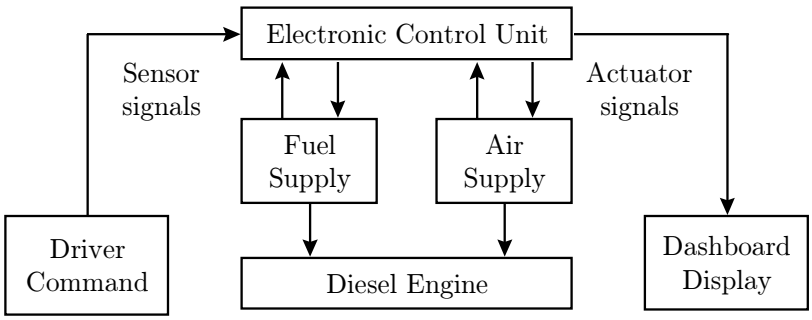

Figure 2.1: Components of the Electronic Diesel Control

achieve the vehicle functions. Automotive structures that combine a physical part and electronic control units are often called *mechatronic systems*. Examples of mechatronic systems are the anti-lock braking system (ABS), the electronic diesel control (EDC), the automatic climate control (ACC) or the vehicle dynamics control (VDC) (see [Bau96]).

Hence, the components of an automotive system can be divided into three main parts: sensors, actuators and the ECU. Sensors measure physical quantities, such as temperature or pressure. These physical quantities are converted into electrical signals and undergo initial conditioning like filtering, amplification, or analog–digital conversion before being processed within the ECU. The ECU then outputs electrical signals that govern actuators in order to achieve the desired functionality.

2.1.1 Example: Electronic Diesel Control

A typical instance of an automotive system is the *electronic diesel control* (EDC) (see [Bau96]). The pedal position sensor considered in section 1.1.1 is a small subsystem of the EDC. The EDC is responsible for supplying appropriate amounts of fuel and air to the diesel engine, in order to provide the thrust necessary for vehicle motion. Figure 2.1 shows a schematic view of this system. The sensors in the EDC provide information about the current condition of the engine and the driver commands. There exist sensors for the following quantities:

- sensors for driver commands: accelerator pedal position, brake pedal switch position and clutch switch position;
- sensors for engine conditions: engine speed, boost pressure, atmospheric pressure, intake air quantity, intake air temperature, fuel temperature and engine temperature.

The actuators in the EDC influence certain input quantities in order to govern the behavior of the diesel engine. Actuators exist for the following quantities:

- actuators for the air supply: wastegate valve position and turbo control valve position;
- actuators for the fuel supply: fuel quantity solenoid, fuel injection pressure solenoid, fuel injection timing solenoid and the fuel shut-off valve;
- indicator lamps to inform the driver in the case of failure.

The electronic control unit (ECU) processes the information from the sensors, performs system-specific control algorithms, and outputs corresponding actuator signals in order to achieve the desired thrust of the diesel engine as demanded by the driver.

2.1.2 Function

The main aim during the design of an automotive system such as the EDC is to ensure that the vehicle will fulfill as much of the designed functionality as possible, even in the case of faults. Engineers define a priority of functions to be fulfilled, such as

- to prevent safety-critical situations (e.g. an uncontrollable engine),
- to prevent severe system damage (e.g. to the engine or gear box),
- to comply with legal restrictions (e.g. regarding emissions, fuel consumption, or noise),
- to maintain mobility (e.g. to have the engine running, possibly with reduced performance),
- to achieve comfort functions (e.g. air condition, smooth running).

2.2 Types of Tasks

During the life cycle of an automotive system such as the EDC, a number of different tasks have to be carried out (figure 2.2). Each of these tasks pursues a different goal and is performed at a different stage within the cycle. However, a common feature of the tasks is that each one requires a certain amount of knowledge about the system. In particular, they require to derive and to analyze information about the system's normal and faulty behavior. In the following, we characterize some of the most important tasks to be carried out.

2.2.1 Failure Modes and Effects Analysis

Failure modes and effects analysis (FMEA) is carried out during the design phase of an automotive system. FMEA aims at investigating and assessing possible causes and effects of system malfunction, in order to pinpoint necessary revisions of the design as early as possible. It is a method for preventive quality assurance

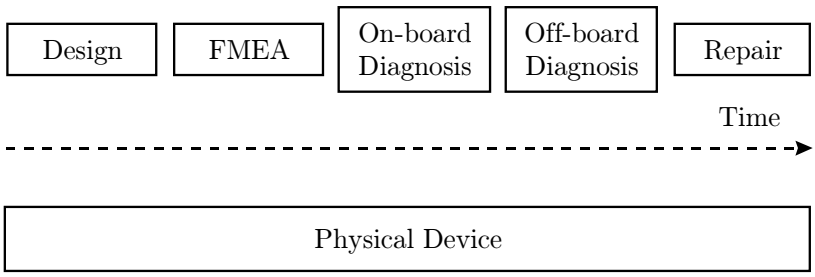

Figure 2.2: Types of tasks during the life cycle of an automotive system

that is of wide-spread use in many companies. The FMEA process starts from the functionality of individual components or processes of the system, and considers possible ways in which these constituents could fail. The possible failures of the constituents are called failure causes. The goal of an FMEA for automotive systems is to predict the effect of the failure causes on the behavior of the subsystem and the whole vehicle. The focus is on the impact of the faults on the functions of the system, such as passenger safety, environmental aspects, and reduction of comfort functions (see section 2.1.2). For instance, during an FMEA of the EDC, the effects of a stuck-at-ground failure of the pedal position sensor switch on the fuel quantity of the diesel engine might have to be considered. The derived behavioral consequences are called failure effects. The priority of functions outlined in section 2.1.2 can be used to assign a severity number to failure effects, ranging from 1 (least severe) to 10 (very severe).

The result of the analysis is documented using three factors: the severity of the failure effect (S), the detectability of the failure (D), and the likelihood of the failure to occur (O). With these three factors, a *risk priority number* (RPN) is obtained as

$$RPN = S \cdot D \cdot O.$$

The magnitude of the risk priority number is the basis for deciding on potential design revisions. The results of the analysis are documented in the form of a standardized table (see figure 2.3). Such a document serves as a certification of the quality of the product, as being demanded to an increasing extent by customers and legal acts.

In order to simplify the analysis, it is common in FMEA to analyze only single faults, to consider only worst case situations, and to document only the differences compared to the normal behavior of the system.

FMEA is carried out by engineers. Since predicting the effect of failures requires knowledge at different abstraction levels, ranging from single component behavior to complex functions of the vehicle, an FMEA team has to involve several people from different departments. FMEA is thus a costly task that binds a lot of valuable resources of the involved engineers.

Component, Process	Function, Purpose	Failure Mode	Failure Effect	Failure Cause	RPN
...	...	...	...	...	...

Figure 2.3: Structure of an FMEA document

2.2.2 On-board Monitoring and Diagnosis

On-board monitoring and diagnosis of automotive systems consists of devising ECU procedures — often based on FMEA results — that continuously supervise the sensor values in order to recognize failures of the system. In case of a failure, alarms have to be signalized in order to warn the driver, or fault codes have to be generated and stored in the ECU memory.

For failures that are considered critical, it might also be necessary to perform so-called *recovery actions*. Recovery actions attempt to take the vehicle back into safe operational conditions again and to partially restore its functionality. Recovery actions can consist of minor performance reductions hardly perceivable by the driver, or so-called limp-home modes that just allow the driver to reach the next service bay, or even a complete shut-down of the engine.

Hence, for on-board diagnosis, localizing a failure without ambiguity is often not essential. Instead, failures have to be identified to a degree where it is possible to choose the right action, i.e. to perform a recovery action, to alarm the driver, or to generate a failure code.

In current practice, on-board monitoring and diagnosis is regarded as an addendum to the design of an automotive system. Current on-board diagnosis procedures of automotive systems detect faults on the basis of pre-defined range and plausibility checks for signals. This means that there is a fixed relationship between violations of plausibility checks, failures that are assumed to cause this and built-in recovery actions that will be performed by the system.

This reflects the fact that at present, the development of on-board diagnosis for automotive systems usually does not follow precisely defined methodologies or criteria. E.g., what is considered to be the normal range of a value or a significant deviation is mostly based on the opinion of experts that design the system and on previous experience, rather than a systematic analysis of the system's behavior and the characteristics of the pursued task.

In control units for the EDC, currently about one half of the software is dedicated to on-board monitoring and diagnosis, and this share is still growing. This reflects also a trend to increase the amount of on-board diagnosis in order to exploit more information about the context of faults that occur only sporadically.

2.2.3 Off-board Diagnosis and Repair

Off-board diagnosis and repair of automotive systems is part of the after-sales and carried out in the field workshops. It aims at re-establishing the complete functionality of an automotive system for which a fault has been detected. This task typically involves the localization and replacement of faulty components.

Hence, the task of off-board diagnosis is to localize failures down to the smallest replaceable unit. In contrast to the other tasks outlined in this chapter, identifying the actual kind of fault is less important for off-board diagnosis. However, it might be relevant for the repair process itself, e.g. in order to suspend tests which could be dangerous to the mechanic.

Off-board diagnosis often uses information from on-board monitoring and diagnosis. The failure codes stored in the ECU can be read out in the service bays, and usually they constitute the starting point for off-board diagnosis. As noted above, however, failure codes as generated by the current control units are not diagnoses in the sense of failure localization. Since they document only violated plausibility checks of ECU signals, it is possible that different component failures lead to the same failure code.

Off-board diagnosis aims at discriminating among them. It is not limited to measuring the ECU signals, but can make use of additional observations, exploiting service equipment ranging from volt meters to motor testers, or even human observations. Due to the standardized and controllable workshop environment, off-board diagnosis can use more information about the application context, e.g. the current driving situation or the load of the engine. It can also devise distinguishing tests, in order to drive the automotive system into a new state and reveal symptoms that where not observable in the original state, e.g. through activating the operating phases of an anti-lock braking system individually.

During off-board diagnosis, a mechanic will usually be guided by repair manuals. Repair manuals are manually created documents that contain diagnostic instructions for a specific car type. They aim at bringing together knowledge from design (e.g., by providing diagrams of the circuit structure) and after-sales (e.g., by ordering test actions according to their costs).

Off-board diagnosis is of great economic importance to car manufacturers. For instance, it is estimated that European passenger cars have an average yearly down-time of 16 working hours due to malfunctions and maintenance. This figure is even greater for commercial vehicles. For the European Community alone, this amounts to a total of over one billion hours for diagnosis and repair. For a large German-American car manufacturer, it is projected that in the future about half the sales will be made with services for the car such as after-sales, and not with the production of the car itself.

2.3 Requirements

The speed-up of innovation cycles and the increased complexity of automotive systems has increased the complexity and costs of all the tasks mentioned above.

In order to understand the fundamental reasons, we have to analyze the challenges that must be faced during these tasks. The most fundamental of these challenges are presented in the following.

2.3.1 Increasing Complexity

With increased environmental awareness, stricter constraints are imposed on the car manufacturers to develop clean cars, and also to keep them clean during their life cycle. For instance, the California Air Resource Board (CARB) mandated that cars sold in the state of California have to be able to monitor certain emission-relevant vehicle subsystems and report faults to the driver ([Cod93]). Also, safety systems such as the anti-lock braking system that provide passive or active protection without effort on the part of the driver are now more and more regarded as part of the standard equipment of a passenger car.

These growing requirements are making the design and maintenance of automotive systems more and more complex. Sometimes, functions can only be achieved through the interaction of several subsystems, which have to communicate data over networks in the vehicle. For instance, the vehicle dynamics control (VDC), a system which assists steering by preventing lateral instability of the vehicle, issues commands to the EDC system and to the braking system in order to reduce the fuel quantity or to increase braking forces whenever it detects a tendency of the vehicle to be pushed out of the turn. Such complex behavior makes manual analysis difficult and error-prone.

2.3.2 System Variants

Automotive systems, like most engineered devices, come in many different variants. They vary from version to version, depending on mounting positions, vehicle type, year of production, and so forth.

One reason is that a supplier of automotive subsystems has to develop his products for many car manufacturers and a lot of car models, all of which impose different requirements on the base system. The actual configuration may differ in the number of sensors and actuators; redundant equipment, such as comfort functions, may be present or absent dependent on the specific car manufacturer. Also, the components themselves come in different constructive details with respect to geometry or material.

This creates a problem for managing and updating the information, but also for behavior analysis and diagnosis of a device at hand. Each of the modifications must be thoroughly handled during fault analysis in FMEA, during the design of on-board diagnosis, and also in the process of off-board diagnosis. Generating specialized solutions for all variants by hand can be infeasible, or is at least very expensive. Thus, there is a need for automated support in order to handle the system variants.

2.3.3 Limited Knowledge

Automotive Systems combine various physical domains. The knowledge about the behavior of certain components, especially components that involve several physical domains like the combustion engine, is incomplete and precarious. In contrast to electric or hydraulic components, there exist no general mathematical models that capture their behavior completely.

Particularly in the case of a failure, certain quantities may not be known exactly or they even remain unknown. For instance, during an FMEA of the EDC, it might be required to predict the effect of a leakage — which causes a too low pressure of the intake air in a diesel engine or a too high exhaust gas recirculation rate — on the emissions and the performance of the vehicle. To some extent, it is therefore necessary to reason about components even if their behavior is ill-specified.

2.3.4 Limited Observability

Automotive Systems have to work in a lot of different situations. The context in which a car is operated in, e.g. as characterized by road and weather conditions or the engine load, is highly dynamic and uncertain. Often, it will neither be measurable for on-board monitoring and diagnosis, nor will it be reproducible for off-board diagnosis in the workshop.

Additionally, in order to achieve cost effectiveness, in general very few sensors are available in automotive systems. For instance, for some versions of the EDC, the hydraulic parts do not contain any sensor at all. The main consequences are incomplete observations and rather qualitative symptom descriptions. However, monitoring and diagnosis have to be capable of processing such information.

2.3.5 Different Objectives

While all of the tasks outlined above require to analyze the behavior of an automotive system, each one is targeted at a different goal. FMEA concentrates on the difference between normal and faulty behavior, in particular behavioral aspects of the output or performance of the system. For on-board diagnosis, the set of actions that can be taken by the control unit determines certain classes of behaviors that need to be distinguished. For off-board diagnosis and repair, the focus is on discriminating faults at the granularity of replaceable components. These different objectives impede a straightforward re-use of results between the different tasks.

2.3.6 Completeness of Analysis

Since they are concerned with safety and environmental aspects, any of the tasks characterized above has to be carried out as completely as possible, particularly for automotive systems like the ABS or the EDC.

For behavior analysis in FMEA, completeness has to be achieved in two respects: it is necessary to cover all the possible failure causes of a system constituent, but also to consider all the relevant circumstances, such as driving conditions, vehicle load, states of interacting subsystems, etc. This makes completeness difficult to achieve, since an enormous number of possible circumstances have to be considered. For diagnosis, covering these situations is a complex task, and often turns out to be infeasible for hand-crafted diagnostic procedures which are based on predefined range or plausibility checks only.

2.3.7 Real-time Requirements

In an on-board environment, there is only limited time available to come up with a diagnostic conclusion. The reason is that in the case of a failure, the automotive system often has to be moved to another state (e.g. shutting down the engine) to prevent safety-critical situations or severe system damage, or to comply with legal restrictions with respect to emissions or noise.

As a consequence, the computational requirements for on-board diagnosis functions are relatively high. On the other hand, the memory and computing resources of current ECUs are relatively low. Thus, one has to focus on essential aspects of behavior only and must avoid any unnecessary detail.

2.4 The Problem

The tasks presented above differ with respect to the phase of the product life cycle during which they can be carried out. They involve different organizational units within a company, or even different companies such as suppliers and manufacturers. As a consequence, the involved knowledge is spread among different places and different points in time.

However, it has been shown that the different task share, at least to a certain extent, common knowledge about the system under consideration. For instance, both in behavior analysis and diagnosis it is essential to know how the system behaves normally, and how it will behave in the case of a failure.

From a work process point of view, it is desirable to integrate different tasks that are concerned with the same type of knowledge. This helps to avoid unnecessary double work and to mitigate update problems that are likely to occur if an automotive system undergoes several revision and modification cycles.

To achieve a more efficient, redundant-free organization of work processes, a common basis for computer-supported representation and utilization of the knowledge underlying FMEA, on-board diagnosis or workshop diagnosis is required. In the following, this will be referred to as *horizontal integration*.

However, horizontal integration is not just a matter of simply storing all the available information in a common database, or writing converters that mediate between different formats and standards. The difficulty lies in the different ways the knowledge is *used*, and in particular the different *granularity* of the knowledge that each of the tasks requires.

For instance, during the design of an automotive system, it might be necessary to use a computer-based simulation of the behavior of its hydraulic subsystem in order to decide on valve diameters, pipe lengths, etc. Later on, during the design of on-board diagnosis of this system, failures of the hydraulic components have to be considered. This also requires knowledge about the behavior of this subsystem, but at a different level of abstraction, as now whole classes of behaviors, such as leakages, occlusions, etc. have to be considered. The simulation model contains knowledge of this kind, but can't be used for this purpose right away, since a specific granularity (i.e., real numbers) and use (i.e., simulating behavior over time) is hard-coded into this piece of work.

As another example, choosing the right level of abstraction is crucial to the task of diagnosis. If diagnostics is based on a model that is inadequate for the task at hand, the diagnostic results are likely to be wrong or useless. A too coarse model may fail to reveal existing contradictions. A too fine-grained model, on the other hand, may suggest inconsistencies that are meaningless in the sense that they rely on insignificant discrepancies of values. Furthermore, using always the best, most accurate and most detailed model available also tends to make the task intractable, or at least unnecessarily complex and resource-consuming.

This illustrates the basic problem of achieving re-use of knowledge at different levels of abstraction. The knowledge, or at least significant parts thereof, might be readily available. But often, it is not represented in a form that would make it amenable for any other task than the one it has been originally used for. It is quite common in the automotive industry that various simulation models exist for different physical domains of a system, but they hardly can be exploited e.g. for on-board diagnosis. However, such a re-use of knowledge becomes more and more important, given the ever-increasing complexity of automotive systems and the emerging trend to achieve more sophisticated functionality through the interaction of several subsystems.

2.5 The Vision

From a more generalized and conceptual point of view, we have to reconcile two opposing objectives, namely generality of knowledge vs. task-specificity of knowledge. The former point is crucial to achieving re-use of knowledge, while the latter point is crucial to achieve efficacy of a certain task.

We propose in this thesis the view that behavior models could form the basis for supporting different tasks along the process chain (figure 2.4). The question is how models of a physical device can be obtained that are adequate for a particular task. We will discuss requirements on the form of such behavior models, and we will present computer-supported methods that deal with the problem of automatically *generating* such behavior models.

In a nutshell, the solution proposed in this thesis (figure 2.5) consists of using a set of generic, re-usable model fragments stored in a library. Task-specificity comes in through the structure of a behavior model, expressed as interconnections

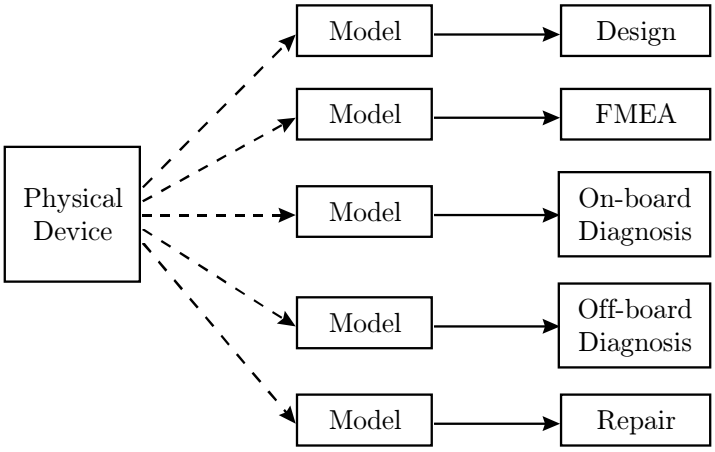

Figure 2.4: Model-based support for different tasks

between instances of these model fragments, and a characterization of the task the model is used for. The granularity of a suitable model is determined both by the structural context and the task requirements, such as the given observability and the need to distinguish certain classes of behaviors from each other.

For instance, in on-board diagnosis, the task might be to discriminate between failures that require different actions. It might be possible or impossible to achieve this for a given structure of an automotive system, given the characteristics of the task in terms of the available sensors, the placement of the sensors, their accuracy, etc.

Essentially, the approach is characterized by "first compose, then transform". A generic model is composed from the model fragments in the library. Transformation operators (see figure 2.5) will be developed that adapt the granularity of a model to a specific task, but preserve the essential information, such that nothing is "lost" compared to the original model. It should be clear that we are particularly interested in transformations that are maximal in the sense that they preserve only the information that is truly essential in order to perform a particular task, and carry no "unnecessary" information. Based on a formal treatment of these informal concepts, we will also be able to investigate to what extent this is possible in principle.

Our approach supports the idea of having a general problem solving method that is independent of the particular model at hand. This contributes to the important separation of general knowledge from the specific ways in which this knowledge is used. It thus helps to achieve the desired horizontal integration and re-use of knowledge.

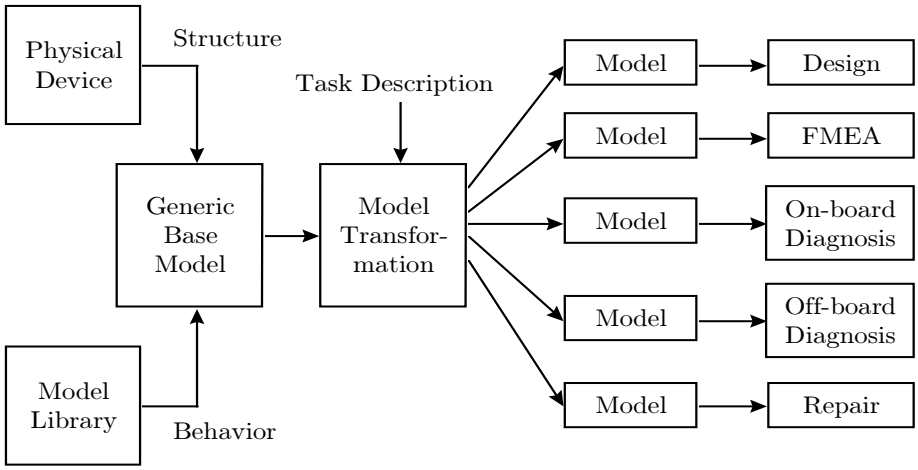

Figure 2.5: Model-based support for different tasks using a generic base model and task-dependent model transformations

2.6 Summary

In this chapter, we have characterized the application domain of automotive systems. It serves as a typical representative of an industrial application, illustrating various tasks and requirements that have to be faced.

A core problem in such a real-world domain is the tension between generality of knowledge vs. task-specificity of knowledge. The former is the prerequisite for re-using knowledge, whereas the latter is a necessary precondition to carry out a task efficiently.

Current practice tends to spatiotemporally distribute the knowledge about the behavior of a system, and thus leads to limited re-use of data that might have been valuable for different tasks. Based on this, we outlined our approach to this problem, which is more general in the sense that it is not limited to the application domain of automotive systems. It consists of composing a base model from generic model fragments, and then to apply task-specific transformations in order to adapt it to the various tasks along the process chain. The next chapter investigates the structure of the required behavior models in more detail.

Chapter 3

Model-based Problem Solving

In this chapter, we are concerned with the representation and use of knowledge that captures the behavior of physical systems. We investigate properties and conceptual features that behavior models must have in order to meet the application requirements presented in chapter 2. To this end, we characterize in a uniform way the different tasks that can be performed with such behavior models.

In order to automate the tasks outlined in section 2.2 such that they can be supported by computer-based tools, we will introduce a formal description of compositional behavior models. This allows to realize the idea of re-using behavioral knowledge by composing a model from a library of model fragments, as presented in section 2.5. We will consider models of components of physical systems that are described as relations, a general concept that covers differential equations, functional dependencies, etc. This formalization serves as a uniform means to represent behavior models, and it is the foundation for subsequent theoretical analysis.

3.1 Model-based Systems

A system that uses an explicit representation of knowledge about the physical world in the form of a model is called a *model-based system*. The question is how and to what extent a model-based system can accomplish the tasks that have been outlined in chapter 2. In particular, we would like to know what roles a device model can play in the various tasks, and what requirements it must fulfill in order to support them. To answer this question, the next section describes how each of the tasks outlined in chapter 2 can in principle be supported by a model-based system (see also [Str00]).

3.1.1 Characterizing Different Types of Tasks

In the following, we characterize how models can be used in order to support the various tasks. In order to keep the description concise, we will do this in an informal way first, and later introduce more precise definitions. In the following,

- $MODEL$ denotes a behavior model (which possibly contains also a description of faulty behaviors),
- $CRITERIA$ denotes a description of certain behaviors (e.g. desired or allowed situations, undesired or not allowed ones), and
- OBS denotes external restrictions on the system (e.g. through observations, specification of operating modes, or specification of parameters).

We will use the informal operators "$\cup$" to denote combinations of the above elements, such as $MODEL \cup OBS$ for a behavior model combined with external restrictions, and "$\vdash$" to denote inference from a behavior model.

Design and FMEA

As it has been described in chapter 2, failure modes and effects analysis (FMEA) tries to predict the consequences of component failures on a device. If we have a model that captures the behavior of the device together with its possible failures, this task can be characterized as the derivation of behavioral descriptions from the model:

$$MODEL \vdash ?.$$

If we focus on FMEA as a means for safety analysis, it could also be understood as checking consistency of the model with certain behavioral descriptions that characterize e.g. safety critical or dangerous behaviors which must be avoided:

$$MODEL \cup CRITERIA \vdash^{?} \bot.$$

This check determines whether certain behaviors can occur. For instance, it could be important to verify that even if one of the return valves of an anti-lock braking system is occluded, it is still possible to brake the wheels of the vehicle.

Monitoring and Diagnosis

In monitoring, we have observations of the system (e.g. sensor signals), and we are interested in knowing whether the device is behaving correctly. Current on-board diagnosis performs this task using a set of pre-defined criteria only, such as plausibility checks or thresholds for signals:

$$OBS \cup CRITERIA \vdash^{?} \bot.$$

However, in general, this task is characterized as checking the consistency of the model with the given observations:

$$MODEL \cup OBS \vdash^{?} \bot.$$

If we have detected an inconsistency — which means that we have detected a failure —, diagnosis amounts to the task of finding the possible origins. To this end, we have to revise the model in such a way that it becomes consistent again with the observations:

$$MODEL \cup OBS \vdash \bot \rightarrow MODEL' \cup OBS \nvdash \bot.$$

The revision procedure, i.e. the step from $MODEL$ to $MODEL'$, can consist of just dropping parts of the model that describe normal component behaviors, or substituting them with descriptions of possible failures. In the former case, the faults will just be localized, whereas in the latter case, also the types of failures can be identified. Hence, a model of faulty behavior will be required for fault identification, but it is not necessarily required for fault localization. Figure 3.1 summarizes the different types of abstract problem-solving tasks.

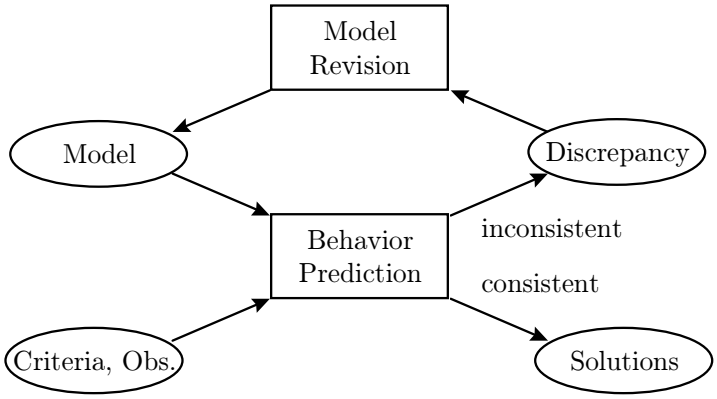

Figure 3.1: Model-based behavior prediction, consistency checking, and model revision

3.2 Relational Behavior Models

Our ultimate goal is to analyze different levels of abstraction of models of a physical system, and to develop methods that automatically transform them. This requires to define behavior models in a strict and formal way.

As we have seen in the previous chapter, behavior models have to express principled knowledge of science and engineering about physical phenomena. Engineers commonly express such knowledge as equations or differential equations over some system variables and parameters, or e.g. as characteristic lines or characteristic maps that have been obtained through evaluating various measurements. Also, manipulation and reformulation of models is often done using this representation, e.g. in order to derive linear approximations, to test if a solution exists, or to solve for output variables of a system. However, representing behavior models directly in the form of symbolic formulae leads to several

difficulties. First, representations for mathematical equations are not unique. It would require to define some normal form, which is difficult even if we restrict ourselves to a subclass of formulae only ([DST93]). Also, using the model would be difficult with such a representation. For instance, determining the admissible values of a variable — i.e., solving for that variable — requires different solution algorithms that depend on the form of the equations, such as Gaussian elimination for linear equations, or Gröbner basis methods for non-linear equations. This conflicts with our goal to have a general problem solving method that is independent of the particular model at hand (section 2.5). While one form might work e.g. for simulation of a model, using the same form for checking if the model contradicts another one, as required e.g. for the diagnosis task, might become intractable or even impossible.

In contrast, many of the AI-centered approaches to modeling device behavior treat models uniformly as sets of logical sentences (e.g. [dKMR92], [Rei87]). This representations allows to prove, for instance, whether a certain model is consistent with another. On the other hand, this level of representation does not reflect the nature of behavior models, as it abstracts from the fact that they capture knowledge about physical phenomena of a device. Hence, this form makes the models hard to handle e.g. for engineers. For instance, there is no obvious way how representations based on logical formulae can be derived from their existing models, which typically correspond to numerical differential equations. What is required is a level of representation that is more amenable to computer-supported methods than symbolic representations of mathematical formulae, but still retains more of the original problem structure than a set of propositional sentences does.

This is the motivation for representing behavior models as relations (also called *constraints*), and operations on behavior models as operations on these relations. Using this representation, we will still be able to talk, in the context of behavior models, about essential concepts such as shared variables, or solutions for a certain variable. The relational representation can thus be viewed as an additional layer in order to mediate and fill the gap between the two extremes characterized above. This is helpful in particular if models of the behavior of physical systems are not constructed from scratch, but stem at least to some extent from the mathematical or engineering level. Also, as we will see later, a relational representation can be used at different levels of abstraction of a model, and thus lends itself to the task of transforming a model.

The relational representation is complete in the sense that it is possible to interpret an arbitrary equation in terms of a relational behavior model. On the other hand, there is a correspondence between relations and logical propositions (see section 3.2.4) and, thus, a well-defined basis for performing inferences, as required e.g. for behavior prediction or diagnosis, on the basis of relational behavior models. Following the considerations above, however, we will regard such logical theories mostly as an optional representation that could be obtained, if necessary, from an underlying relational model. In the following, we will present the foundations of relational behavior models.

3.2.1 Relations

Relations capture behavior information by restricting the set of possible behaviors. This is based on the view that the behavior of a device can be described in terms of an ordered set of variables

$$\boldsymbol{v} = (v_1, v_2, \ldots, v_n).$$

By $\boldsymbol{v}[v_{i_1}, v_{i_2}, \ldots, v_{i_k}]$ we denote a restriction of $\boldsymbol{v}$ to a subset of k variables $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$. The term variable refers to internal variables, output variables, state variables, but also derivatives thereof, or time. Each of the variables v_i, where $i = 1, 2, \ldots, n$, can take values from a domain $DOM(v_i)$:

Definition 1 (Domain) *A domain, denoted DOM, is a set of values which are defined extensionally (i.e., by enumerating them) or intensionally.*

A domain can consist of an infinite number of elements, e.g. the real values, integers or intervals, but also a finite number of elements like a set of symbolic values, or the "left" and "right" states of the idle switch in the example considered in section 1.1.1. It can be ordered, partially ordered, or not ordered at all. For v_i and v_j, $DOM(v_i)$ and $DOM(v_j)$ are not necessarily the same, i.e. each of the variables used to describe the behavior of a device can possibly have a different domain. Thus, the cartesian product

$$DOM(\boldsymbol{v}) = DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$$

denotes the possible space to describe the behavior of a device. A relation type constrains the set of possible value combinations:

Definition 2 (Relation Type) *A relation type (or constraint type) RT with arguments* $\boldsymbol{arg} = (arg_1, arg_2, \ldots, arg_k)$ *denotes a subset of the possible behavior space:*

$$RT(\boldsymbol{arg}) \subseteq DOM(arg_1) \times DOM(arg_2) \times \ldots \times DOM(arg_k).$$

Relations (or constraints) can be derived from a relation type by identifying its arguments with a subset of the variables:

Definition 3 (Relation) *Let RT be a relation type with arguments* $\boldsymbol{arg} = (arg_1, arg_2, \ldots, arg_k)$. *Let* $\boldsymbol{v}$ *be variables such that* $DOM(v_{i_1}) = DOM(arg_1), \ldots, DOM(v_{i_k}) = DOM(arg_k)$. *A relation (or constraint) R is the result of identifying the variables* $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ *with arguments* $\boldsymbol{arg}$ *in* $RT(\boldsymbol{arg})$*:*

$$R(v_{i_1}, v_{i_2}, \ldots, v_{i_k}) = RT(\boldsymbol{arg}).$$

An element of a relation, $\boldsymbol{val} \in R$, is termed a *tuple*. The ordered set of variables that a relation R is defined on is called the *scheme* of R and denoted $scheme(R)$. Relations that are based on the same relation type consist of the same set of tuples, but can have a different scheme. If it is not clear from the context, we write $R.v_i$ to make explicit that a certain variable v_i belongs to $scheme(R)$.

3.2.2 Basic Operations on Relations

In order to use relations for capturing behavior — e.g. deriving the consequences of observing physical situations of a device, or connecting several device constituents to form a larger device — we have to define operations on relations. In the following, we define three basic operators on relations: projection, selection and join. They form the basis for manipulations of relational behavior models in the sense that more complex operations on models can be expressed in terms of these basic operations. These operators correspond to operations known in relational algebra ([Ull89], [Fre93]). However, we are also interested in the interpretation of the operations with respect to the underlying mathematical level.

Definition 4 (Projection) *Let $R(\boldsymbol{v}) \subseteq DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$ denote a relational behavior model. Let proj denote a subset of variables of $\boldsymbol{v}$. Then the projection of R on proj, denoted $\Pi_{proj}(R)$, is defined as*

$$\Pi_{proj}(R) := \{\boldsymbol{val}[proj] \mid \boldsymbol{val} \in R\}$$

From the mathematical view, the projection operation on a subset of model variables corresponds to the elimination of variables. A special case is to solve a set of equations for a single variable by eliminating all the other variables.

Definition 5 (Selection) *Let $R(\boldsymbol{v}) \subseteq DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$ denote a relational behavior model. Let cond denote a selection criterion for tuples of R. Then the selection of R witch respect to cond, denoted $\sigma_{cond}(R)$, is defined as*

$$\sigma_{cond}(R) := \{\boldsymbol{val} \in R \mid cond(\boldsymbol{val}) \text{ is true}\}$$

In particular, *cond* can be the restriction of variables to a subset, or single elements, of their domain. The selection operation corresponds to restricting a model (or mathematical formulae) through some "external" information, e.g. observations of input variables, or specification of parameters.

Definition 6 (Join) *Let $R(\boldsymbol{v}) \subseteq DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$, $S(\boldsymbol{w}) \subseteq DOM(w_1) \times DOM(w_2) \times \ldots \times DOM(w_m)$ denote two relational behavior models. Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_k)$ denote the subset of variables common to $\boldsymbol{v}$ and $\boldsymbol{w}$, i.e.*

$$\boldsymbol{u} = scheme(R) \cap scheme(S).$$

Then the join of R and S, denoted $R \bowtie S$, is defined as

$$R \bowtie S := \Pi_{scheme(R) \cup scheme(S)} \sigma_{(R.u_1 = S.u_1) \wedge \ldots \wedge (R.u_k = S.u_k)}(R \times S).$$

Joining corresponds to combining sets of mathematical formulae through the identification of variables, which can lead to additional restrictions of the other variables.

3.2.3 Combining Relations to Networks of Relations

Using the operators in section 3.2.2, relations can be implicitly defined using sets of smaller relations, just as composing sets of equations:

Definition 7 (Constraint Network) *A constraint network over variables $\boldsymbol{v}$ is a set of relations $R_1, \ldots, R_m$, where for each relation R_i, $scheme(R_i)$ is a restriction of $\boldsymbol{v}$, and $\bigcup_i scheme(R_i) = \boldsymbol{v}$. The constraint network implicitly represents a unique relation R with $scheme(R) = \boldsymbol{v}$, which stands for all consistent assignments to the variables:*

$$\begin{aligned} R &:= \{\boldsymbol{val} = (val_1, \ldots, val_n) \mid \forall R_i : \Pi_{scheme(R_i)}(\boldsymbol{val}) \in R_i\} = \\ &= R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m. \end{aligned}$$

A constraint network is said to define a *constraint satisfaction problem* (CSP). The tuples of the relation R defined by the constraint network are called the solutions of the CSP. A *binary* CSP is a special case of constraint satisfaction problem where for all relations R_i, $scheme(R_i)$ consists of at most two variables.

3.2.4 Interpretation of Relations as Propositional Theory

As noted during the beginning of this section, many of the early approaches to model-based problem solving are based on representations of device behavior in terms of logical sentences (e.g. [Rei87]). In fact, there exists a close relationship between relational models as presented above and logical theories. Given a relation R, a propositional formula Φ can be constructed such that each model of Φ corresponds to a tuple of R:

Definition 8 (Propositional Encoding) *Let R be a relation. Let the proposition $v_i = val$ denote that variable $v_i \in scheme(R)$ takes the value $val \in DOM(v_i)$. The propositional encoding of R is the formula Φ defined by*

$$\begin{gathered} \forall v_i : \exists val \in DOM(v_i) : v_i = val \\ \wedge \forall v_i : (v_i = val_1 \in DOM(v_i) \Rightarrow \forall val_2 \in DOM(v_i) \setminus \{val_1\} : v_i \neq val_2) \\ \wedge \forall \boldsymbol{val} = (val_1, val_2, \ldots, val_n) \notin R : \neg(v_1 = val_1 \wedge \ldots \wedge v_n = val_n). \end{gathered}$$

The formula Φ expresses the fact that each variable must take a value from its domain, the fact that these values are pairwise exclusive, and the possible value assignments allowed in R. Φ is a "negative" encoding of R in the sense that it represents only the tuples that are not allowed in R.

It holds that every model of the propositional encoding of R corresponds to a tuple of R. In particular, R is non-empty if and only if Φ has a model. This provides a basis for interpreting the manipulation of relational models as deriving inferences in propositional logic, and vice versa. The relationship between constraint systems and logic is investigated in more detail in [Mac92].

3.3 Conceptual Modeling

In this section, we investigate the impacts of the tasks and their requirements on the employed behavior models in more detail. As has been shown in section 3.1.1, model-based problem solving tasks require not just deriving inferences from the model, but also changing (i.e., revising) the behavior model itself. This is even true for a prediction task such as FMEA, since it deals with the effects of different faulty behaviors of components in exchange for their normal behavior. Accomplishing this efficiently imposes requirements on the form of the model. For instance, in FMEA, predicting failure effects for the different failure models of a component model would be very inefficient, if the rest of the model had to be built from scratch each time. In diagnosis, checking whether certain failure hypotheses are consistent with the observations would be infeasible if it involved re-building of the complete model.

This illustrates that we need to organize behavior models in such a way that they support model revision, in the sense that changes can be performed locally, and unaffected parts of the model (and consequently, unaffected inferences) can be re-used for the revised model.

This can be achieved by a meta-concept for behavior models that further structures a model into two parts: a set of model fragments and a structural description. The model fragments capture the behavior of basic constituents of the device. The structural description defines the current instances of model fragments and the way they interact with each other. Thus, a behavior model of a device can be composed from a structural description and a library of model fragments. This separation promises efficient solutions for behavior prediction and diagnosis, since part of the model (and inferences) can be re-used during model revisions. It is also the foundation to remedy the variant problem outlined in section 2.3.2, because different structural variants of a device involve modifications of the structural description only, whereas the model fragments in the library remain the same.

3.3.1 Ontologies for Conceptual Modeling

To put this idea to work, it is necessary to develop a conceptual layer for behavior models that deals with the representation, composition and revision of models at the level of model fragments. In AI, such conceptual layers are termed *ontologies*. There exist two major types of ontologies, which essentially differ in the extent and type of possible model revisions that they are able to handle.

In process-oriented ontology ([For90]), the basic behavior constituents of a model are processes. Processes describe interactions among objects, such as a heat flow between a flame and liquid in a container. The view is that the behavior expressed in the process model cannot be allocated to one of the involved objects alone, but is rather due to a certain relationship between the objects (such as spatial proximity) and certain preconditions (such as the presence of a temperature difference between the objects). Whenever the conditions for their

activity are true, processes will come into existence. In doing so, one process can also create the necessary preconditions for another process (such as a boiling process, if the temperature of the liquid exceeds a certain limit). Conversely, processes cease to exist if the conditions for their activity are invalidated (such as a balanced temperature, or the container being removed).

Thus, process-oriented modeling is suited to describe the behavior of dynamic systems that lack a fixed a priori structure of constituents, like chemical processes or ecosystems. It has been applied e.g. to thermodynamic systems ([For90]), waste water treatment ([SH98]) and the prediction of algae blooms ([HS97]). One can say that the higher the degree of model revision, the more important the ontological level becomes and the more sophisticated is has to be. Hence, it is not surprising that the high flexibility of a process-oriented description comes at a considerable complexity for composing and diagnosing such models ([Hel01]).

Automotive systems, on the other hand, are highly structured devices. Since modularity simplifies the design of an engineered device considerably, there often exists a natural decomposition of the device into a limited number of component types, such as valves, relays, resistors, etc. The instances of these component types may differ with respect to some parameters (e.g. a valve diameter), but they share the same type of behavior and the same paths of interaction (e.g. two hydraulic ports and a command line). Consequently, the behavior of a device can be obtained as the interaction of a fixed set of component type instances. From the point of view of diagnosis, faulty behaviors are due to failures of one or more components. These failures can be described by failure models which are also local to the component types.

These considerations are the reason why we focus on a component-oriented ontology ([Dav84, dKB90]) for modeling devices such as automotive systems. The component-oriented ontology describes the behavior of a device in terms of components and conduits that define possible paths of interactions among them. Thus, on the one hand, component-oriented ontology has the desired feature to enable the localization of behavior descriptions in the form of model fragments. On the other hand, compared to process-oriented modeling, it is a reasonable compromise in terms of the number of the concepts at the conceptual layer that need to be introduced. It is worth noting that the component-oriented ontology can be cast as a special case of the more general process-oriented ontology (see [SH98, Hel01]).

Diagnostic applications of component-oriented modeling have focussed mostly on devices that involve simple logical gates or electrical circuits ([dKW87]). It is appropriate when other properties of the "stuff" transported via the conduits — like binary signals, electrical current or voltage — can be ignored, and no significant "stuff" is stored within conduits. As will be discussed later, however, this implicit assumption can present a limitation in the context of real automotive systems, in particular when dealing with phenomena like the fuel combustion process or the formation of exhaust emissions.

3.4 Component-oriented System Descriptions

We can now augment our basic notion of relational behavior models (section 3.2) by primitives that further structure them at a component-oriented, conceptual level. As outlined above, the model is separated into a structural part and behavioral fragments:

- *component types* are physical constituents of a device (such as valves, relays and resistors) that share the same normal and faulty behavior and the same interaction paths.
- *conduits* (or *connections*) are paths that pass material or information between component types. This is the only way by which component types are allowed to interact with their environment.

A model that is composed of a behavioral part describing the behavior of constituents and a structural part describing the structure of the constituents is called a *system description* (see also [Rei87]).

3.4.1 Behavioral Part

A behavioral model fragment captures the physical phenomena associated with a component type in terms of variables that characterize its operation:

Behavior Mode

A behavior mode, denoted *mode*, captures a distinct behavior (e.g. Ohm's law), which is common to all instances of a component type. The values of *mode* are classified as either correct or faulty, e.g. "ok" or "shorted" for the case of a resistor. Changing the behavior mode is thus the starting point for revising a model during diagnosis. There is a default behavior mode which is considered valid if no revision of the model has yet taken place. The behavior modes can also have associated probabilities.

Behavior Description

A behavior description for a component type specifies a set of relation types among the variables of a component type that captures its behavior for a particular behavior mode. It is possible that only the correct behavior mode has an associated relation. In this case, the faulty behavior is unrestricted, i.e. equal to $DOM(\boldsymbol{v})$. The variables used in a behavior description can be further classified into internal variables and external variables.

Internal Variables

Internal variables are visible only to the component itself, and cannot be accessed by other components. There are three different types of internal variables:

- *parameters* are constants that describe attributes of specific instances of a component type, e.g. the resistance of a resistor component type, or the opening diameter of a valve component type.

- *state variables* are dynamic internal variables which can change over time and represent operational states of a component, e.g. the state of a switch or the filling level of a tank.

- *intermediate variables* are variables local to a behavior description. They are introduced only for convenience, e.g. in order to further structure complex behavior descriptions by defining intermediate expressions.

The behavior mode could also be viewed as a distinct internal variable that switches between different behavior descriptions, with the difference that it is subject to external revision procedures.

External Variables

External variables, also called interface variables, are exogenous to a component and can be shared with other component models. Thus, they describe quantities through which a component type can interact with its environment, e.g. electrical current or air temperature. A *terminal* is a collection of external variables in order to describe a physical interconnection, e.g. an electrical terminal comprising interfaces for current and voltage, or a hydraulic terminal comprising interfaces for pressure, flow and temperature.

3.4.2 Structural Part

The structural part of the system description defines the instances $COMPS$ of component types in a behavior model and the way they are interconnected. Since components can only interact through their terminals, the latter is achived by a definition of shared terminals. A structural description could be e.g. the output of a CAD system during the design phase of a device.

The separation of a structural part and a behavioral part in a system description enables the separation of general-purpose knowledge in the form of knowledge about component type behavior and device-specific knowledge in the form of a description of the device structure, as demanded in section 2.5.

Example 1 (PPS Model) *A system description can be defined to capture the behavior of the pedal position sensor (section 1.1.1). The behavioral part consists of five component types:*

- *Switch with external variables:* v_{left}, v_{right}, v_{switch}, *position*
  Internal variables: switchstate (state variable), $pos_{switching}$ *(parameter)*
  Behavior description:

$$switchstate = left \Rightarrow v_{switch} = v_{left},$$
$$switchstate = right \Rightarrow v_{switch} = v_{right},$$
$$position \leq pos_{switching} \Leftrightarrow switchstate = left,$$
$$position > pos_{switching} \Leftrightarrow switchstate = right$$

- *Potentiometer with external variables:* v_{left}, v_{right}, v_{pot}, $position$
  Behavior description:
  $$v_{pot} \cdot 100\% = (100\% - position) \cdot v_{left} + position \cdot v_{right}$$
- *Node with external variables:* v_1, v_2, v_3
  Behavior description:
  $$v_1 = v_2 = v_3$$
- *Battery with external variables:* v_{batt}, v_{gnd}
  Internal variables: batt (parameter), gnd (parameter)
  Behavior description:
  $$v_{batt} = batt, v_{gnd} = gnd$$
- *Pedal with external variables: angle, position*
  Behavior description:
  $$position = angle$$

Assume there is only one behavior mode per component type, and that the domain is

$$DOM = \{[0V, 2V), [2V, 4V), [4V, 6V), [6V, 8V), [8V, 10V)\}$$

for variables involving voltage and

$$DOM = \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$$

for variables involving position and angle, where 0% *means that the gas pedal is in rest position, and* 100% *means that the pedal is fully pushed through. Let the domain for position times voltage (occuring in the potentiometer model) be given as a domain with* 20 *elements*

$$DOM = \{[0, 50), [50, 100), \ldots, [950, 1000)\}.$$

The structural part of the system description contains two instances of the component type Node (denoted $Node_1$ *and* $Node_2$*) and one instance of the other component types (denoted Switch, Potentiometer, Pedal, Battery). The connectivity of these six components is described by*

$$Pedal.position = Potentiometer.position = Switch.position,$$
$$Potentiometer.v_{left} = Node_1.v_3,\ Potentiometer.v_{right} = Node_2.v_3,$$
$$Switch.v_{left} = Node_1.v_2,\ Switch.v_{right} = Node_2.v_2,$$
$$Node_1.v_1 = Battery.v_{gnd},\ Node_2.v_1 = Battery.v_{batt}$$

□

3.5 Model Composition

Model composition denotes the process of mapping a conceptual model to the underlying mathematical relation. The behavior descriptions associated with the conceptual elements, i.e. the relation types for the specific behavior modes of the components, are given in the behavioral part of the system description. During the model composition step,

(1) the conceptual elements $C_i \in COMPS$ have to be instantiated with their corresponding behavior descriptions (i.e. their relation types are instantiated to form relations RC_i),

(2) potentially, the behavior descriptions have to be further refined by instantiating them with with specific parameters or modes,

(3) the structural part of the system description has to be translated into a definition of shared variables for the relation schemes.

Thus, the result of model composition is a constraint network (see section 3.2.1), which forms the basis for prediction or consistency checking. Figure 3.2 provides a schematic view of the model composition step.

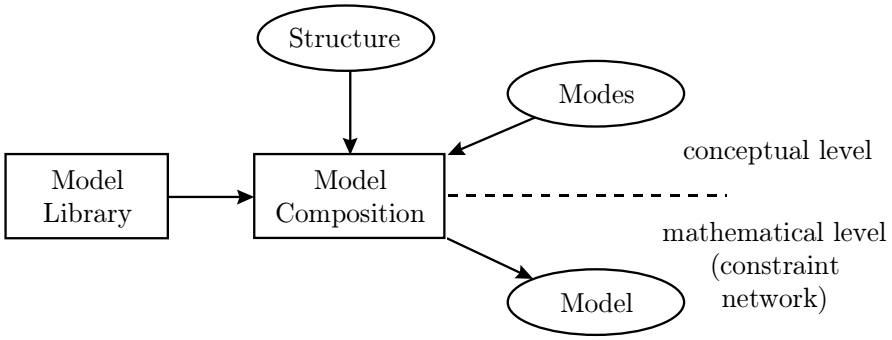

Figure 3.2: Model composition maps a behavior model from the conceptual level (top) to the mathematical level (bottom)

Model composition requires that the behavior constituents contain only local behavior descriptions, which do not make assumptions about other behavior constituents or global properties of the system. Otherwise, it could not be guaranteed that each conceptual model can be mapped to a mathematical model. This is called the *no-function-in-structure* principle ([dKB90]). It states that, for instance, a model of an outlet valve must not assume a specific direction of the flow, even if this should cover all the situations that occur during the normal behavior of the device. If it becomes necessary for other components to change their behavior description, e.g. due to a leakage failure, the valve model would

cause an inconsistency at the mathematical level. Hence, violation of the no-function-in-struture modeling principle can significantly reduce the utility and re-usability of a model fragment.

The model composition step is more or less trivial for a component-oriented ontology. However, it can be more difficult to perform for process-oriented ontologies, since in this case the existence of one behavior constituent, i.e. process, can render the preconditions for the existence of other processes invalid, which means that model composition becomes non-monotonic ([For90, SH98, Hel01]).

3.6 Performing Problem Solving Tasks

The following two sections describe how prediction and consistency checking can be performed on the basis of a composed model.

3.6.1 Model-based Prediction

The constraint network that is the output of model composition implicitly describes all the possible states of the model. On the mathematical level, the states are equivalent to the solutions of the constraint network:

Definition 9 (State) *A state is a tuple of the relation*

$$RC_1 \bowtie RC_2 \bowtie \ldots \bowtie RC_m, C_i \in COMPS,$$

where RC_i is the relation for component C_i, and $COMPS$ are the components in the system description.

External restrictions R_{ext} on a subset of the variables in $\boldsymbol{v}$, such as observations, further restrict the set of possible states:

Definition 10 (External restriction) *An external restriction is a relation*

$$R_{ext} \subseteq DOM(\boldsymbol{v}).$$

Given an external restriction R_{ext} that reduces the possible states of the model, the task of model-based prediction is to determine the remaining states, i.e. tuples of the relation

$$SOL(\boldsymbol{v}) = R(\boldsymbol{v}) \bowtie R_{ext}.$$

To this end, the constraints corresponding to the external restrictions and the constraint network resulting from the model composition step have to be used to derive restrictions on the remaining variables. Model-based prediction is also called *state completion*, since the initial restrictions R_{ext} can be viewed as a partial state description, and the determination of possible states as the task of completing it by the means of the model. State completion does not

involve the conceptual level of the model. Hence, this step is not specific to model-based problem solvers. There exists a variety of algorithms in order to derive solutions of a constraint satisfaction problem (see e.g. [Tsa93]). Due to the inherent complexity of the problem, model-based predictors are often based on incomplete constraint satisfaction algorithms, which means that they cannot rule out all inconsistent states. Consequently, they will derive only a superset of the possible states $SOL(\boldsymbol{v})$.

In the case of finite domains, the resulting states can be explicitly enumerated. This is called an envisionment of the system ([dKB90]). However, for larger systems, the envisionment can be overwhelming. In these cases, it is preferrable to have an implicit characterization of the states only. This is captured by an orthogonal solution:

Definition 11 (Orthogonal Solution) *An orthogonal solution characterizes a set of possible states by independent restrictions for variables:*

$$SOL'(\boldsymbol{v}) = \Pi_{v_1}(SOL(\boldsymbol{v})) \times \Pi_{v_2}(SOL(\boldsymbol{v})) \times \ldots \times \Pi_{v_n}(SOL(\boldsymbol{v})).$$

An orthogonal solution characterizes states by individual restrictions for variables. Compared to sets of states, the expressive power is limited, since disjunctions of tuples (like e.g. the set {(0,0),(1,1)}) cannot be expressed. However, it can be computed by efficient algorithms and is sufficient for most purposes. In particular, the set of solutions is empty if and only if the set of orthogonal solutions is empty.

Example 2 (Model-based Prediction for PPS Model) *Consider again the behavior model given in example 1. Let the parameters for the components Battery and Switch be specified as*

$$gnd = [0V, 2V),\ batt = [8V, 10V),\ pos_{switching} = 40\%.$$

For an external restriction that restricts variable Pedal.position to 20%*,*

$$\Pi_{Switch.state}(SOL(\boldsymbol{v})) = \{left\}, \Pi_{Switch.v_{switch}}(SOL(\boldsymbol{v})) = \{[0V, 2V)\}.$$

Conversely, for an external restriction that restricts variable Switch.state to the value $left$,

$$\Pi_{Pedal.position}(SOL(\boldsymbol{v})) = \{0\%, 20\%\}.$$

□

The above example illustrates that model-based prediction has no built-in "direction" for evaluation, as opposed e.g. to numerical simulation models.

3.6.2 Model-based Diagnosis

Model-based diagnosis also starts from initial restrictions R_{ext} and the constraint network that is the output of model composition. In the context of diagnosis, the restrictions R_{ext} typically correspond to observations OBS of the system.

Diagnosis consists of two tasks. The first is to check consistency of OBS with the model, and the second is to revise the model in the case of an inconsistency. The former task corresponds to checking whether consistent states exist on the level of the mathematical relations, i.e. if it holds that

$$SOL(\boldsymbol{v}) = \emptyset.$$

This can be accomplished by model-based prediction as outlined in the previous section. Model-based prediction can therefore be viewed as a precondition for model-based diagnosis.

The second task, the revision of the model in case of an inconsistency (i.e. if $SOL(\boldsymbol{v})$ is empty), aims at re-establishing consistency of the model with the observations. As noted earlier, this involves the conceptual layer of the model.

In a component-oriented ontology, the structure of the device is assumed to be fixed. Thus, a revision step consists of changing the behavior mode of one or more components. Hence, the search space for diagnostic revisions is determined by the components and their possible behavior modes. An element of this search space is called mode assignment:

Definition 12 (Mode Assignment) *Let $COMPS' \subseteq COMPS$ be a set of components. A mode assignment, denoted MA, is an assignment of behavior modes to components:*

$$\bigwedge_{C_i \in COMPS'} RC_i.mode = m_i \text{ for } m_i \in DOM(RC_i.mode).$$

MA is called complete if $COMPS' = COMPS$, and partial otherwise.

A diagnosis can be characterized as a mode assignment that re-establishes consistency of the model with the observations:

Definition 13 (Consistency-based Diagnosis) *Let MA be a complete mode assignment. MA is called a consistency-based diagnosis for a behavior model $R(\boldsymbol{v})$ and a set of observations OBS, if*

$$\sigma_{MA}(R(\boldsymbol{v})) \bowtie OBS \neq \emptyset.$$

MA is minimal if no mode assignment MA' that substitutes fault modes occuring in MA for correct modes is a consistency-based diagnosis.

Consistency-based diagnoses are sometimes termed candidates, because further observations during the diagnostic process could either confirm or refute

a diagnosis. From the perspective of the characterization in section 3.1.1, a consistency-based diagnosis serves the purpose of two task, namely failure localization and failure identification.

If we are only interested in localization of failures, it is not necessary to know the failure mode for each of the components. Instead, if

$$ok_i \subseteq DOM(RC_i.mode)$$

denotes the correct behavior modes of a component C_i, it is sufficient to know if $RC_i.mode \in ok_i$ or $RC_i.mode \notin ok_i$ holds. This leads to a weaker form of diagnosis, which has been first described in [dKW87], [Rei87] and is defined as follows:

Definition 14 (Consistency-based Fault Localization) *Let $\Delta \subseteq COMPS$. Δ is a consistency-based fault localization for $R(\boldsymbol{v})$ and OBS, if*

$$\sigma_{FL}(R(\boldsymbol{v})) \bowtie OBS \neq \emptyset$$

where FL is defined as

$$\bigwedge_{C_i \in \Delta} RC_i.mode \notin ok_i \wedge \bigwedge_{C_i \in COMPS \setminus \Delta} RC_i.mode \in ok_i.$$

A consistency-based fault localization Δ is minimal if no proper subset $\Delta' \subset \Delta$ is a consistency-based fault localization.

Consistency-based fault localization has been implemented in a system called General Diagnosis Engine (GDE) ([dKW87]). De Kleer and Williams ([dKW87]) have also laid the foundation for efficient computation of consistency-based diagnoses based on so-called *conflicts*. Conflicts capture certain parts of the model that give rise to an inconsistency. More precisely, conflicts describe (partial) mode assignments that cannot hold:

Definition 15 (Conflict) *Let MA be a mode assignment such that*

$$\sigma_{MA}(R(\boldsymbol{v})) \bowtie OBS = \emptyset.$$

Then MA is called a conflict. MA is minimal if no mode assignment MA' that restricts MA to a proper subset of its components is a conflict.

It follows that at least one of the behavior modes involved in a minimal conflict has to be changed in order to re-establish consistency. Conflicts thus provide a starting point for model revision that is more focused than just arbitrarily choosing behavior modes of components for revision. Examples for conflicts and diagnoses will be given in chapters 7 and 8.

It has been shown in [dKW87] that if the conflicts involve only correct behavior modes, the set of minimal conflicts characterizes all possible consistency-based

diagnoses for a model and OBS. Technically, the computation of diagnoses is thus reduced to the computation of minimal conflicts.

Since conflicts link elements of the conceptual level (mode assignments) to properties of the mathematical level (consistency), computing them requires a link between the conceptual level and the constraint network. Intuitively, during processing steps at the mathematical level, it is necessary to keep track of the information which conceptual elements are involved at each step.

De Kleer ([dK86]) proposes an assumption-based truth maintenance system (ATMS) that accomplishes this task. An ATMS logs the elements of the conceptual layer (i.e. the assignments of behavior modes) that are used during the solving process of the constraint network in the form of labels for inferences. Actually, this can be viewed as a more general task, and it is independent of the goal of model-based diagnosis. The labels can be used to derive conflicts, but they are also useful in order to determine which inferences remain valid and thus can be re-used if the model has undergone certain changes on the conceptual layer.

From a practical point of view, it might turn out to be difficult to exhaustively describe the faulty behaviors of a component. However, if the faulty behavior is unknown (i.e., the corresponding relation is unrestricted), nothing useful can ever be inferred from a component assuming this mode. One way to remedy this is to assign failure probabilities to fault models, and to assign the lowest probability to behavior modes that represent unknown faults. This is the basis of a system called $SHERLOCK$ which has been described in [dKW89]. It allows a ranking of mode assignments according to their probability, but still, it provides no means to prove the correctness of components, since an unknown fault can never be refuted.

Deriving correctness of components is useful because it can help to improve fault localization. The principle is that if the fault models are assumed to be complete, and if all fault models of a component have been refuted, then the correct behavior mode must hold for this component. Hence, the component can be removed from the set of candidates, which helps to constrain the set of possible diagnoses. This principle is called physical negation and has been realized in the GDE+ system ([SD89]). It requires to make an assumption about completeness of the set of fault models. An alternative approach to achieve the same effect is to characterize the physically impossible behaviors for a component ([FGN90]).

However, it turns out that if the fault modes are assumed to be complete, minimal conflicts are no longer sufficient to characterize the set of all diagnoses ([dKMR92]). For this case, [dKMR92] present an alternative characterization of fault localizations based on so-called kernel diagnoses. For fault identification, [DS94] provides a characterization that is based on default logic ([Rei80]). Figure 3.3 provides an overview of the framework for model-based problem solving.

3.6.3 Trading off Diagnosis against Prediction

The tasks of prediction and diagnosis are interrelated in the sense that it is possible to trade the number of model revision cycles to be carried out (figure 3.3) for

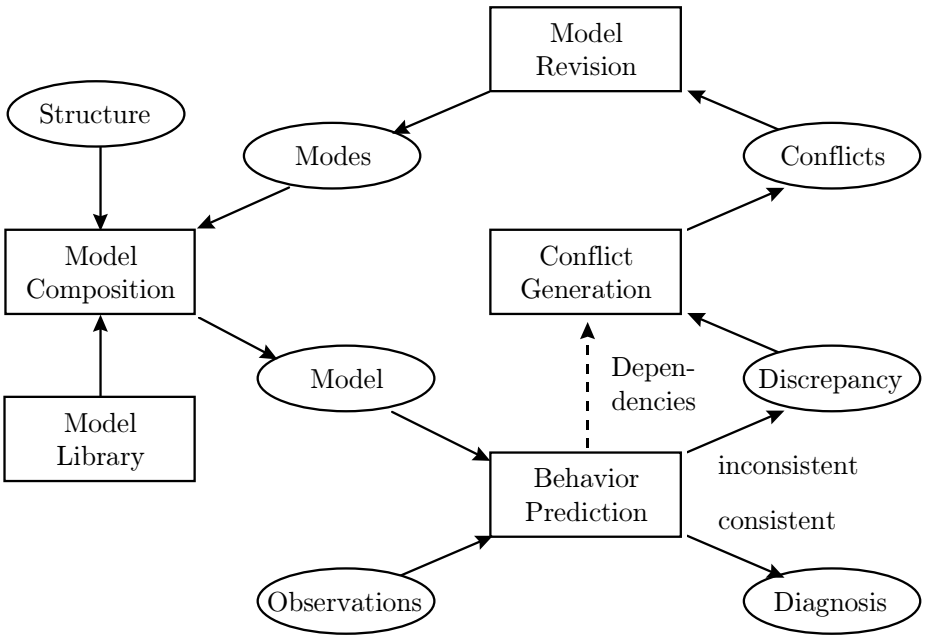

Figure 3.3: Basic elements of a model-based problem solving architecture (see also [SSC00b])

the size of the involved constraint network. The idea is that if possible revision steps are known a priori, they can be anticipated in the constraint network, such that it is sufficient to "switch" between different parts of an extended constraint network instead of modifying it in order to establish consistency with observations.

For a component-oriented ontology, the model revisions performed during diagnosis are limited to changing the behavior modes of components. Hence, it is possible to anticipate the model revision steps by including the behavior models of different behavior modes in the initial constraint network. This can be accomplished by extending the constraint network to include some or all of the behavior modes of a component:

$$R(\boldsymbol{v}) = \sigma_{s_1}(RC_1) \bowtie \sigma_{s_2}(RC_2) \bowtie \ldots \bowtie \sigma_{s_m}(RC_m)$$

where s_i specifies a subset of the behavior modes of a component C_i. More generally, it is possible to define a spectrum between prediction and diagnosis, depending on how much revision steps are included in the constraint network. In the extreme, if every possible revision is anticipated in the model, the constraint network completely covers all situations that can occur, and consequently, there will never by an inconsistency of the constraint network with external restrictions. Thus, no revision of the model is necessary, and diagnosis is reduced to prediction using the extended constraint network. This is the basis of approaches that cast model-based diagnosis as a constraint satisfaction problem (e.g. [El 98]).

However, anticipating possible model revisions comes at certain costs. The more revision cycles are omitted, the larger the size of the required constraint network. Furthermore, as the size of the constraint network increases, it is more and more likely that parts of the constraint network will actually never be used during problem solving. Trading model revision for the size of the constraint network is thus more difficult for a process-oriented ontology. Since in this case, the possible model revisions to be anticipated are not limited to the change of behavior modes, their number can be infeasibly large.

3.7 Discussion

What have we gained by now, in terms of the challenges outlined in the previous chapter? Compared to traditional diagnostic approaches that are based e.g. on pre-defined plausibility checks of variables, the outlined approach of consistency-based diagnosis offers a number of advantages:

(1) *Diagnosis of multiple failures:* The set of consistency-based diagnoses — or kernel diagnoses — characterizes all possible solutions in the diagnostic search space. This covers also combinations of failures, which is often beyond the capabilities of hand-made analysis.

(2) *Completeness of the results:* The previous point is also the reason why it is possible to guarantee completeness of the result, provided that the model of the device is accurate. This was one crucial requirement in order to achieve the necessary coverage when dealing with safety-critical systems.

(3) *Models of correct behavior suffice:* It has been shown that in order to perform consistency-based fault localization, only models of the correct behavior of a device are necessary. This means that unanticipated failures can be diagnosed, which is an advantage compared e.g. to pre-defined plausibility checks.

We have also seen that a compositional model can be easily adapted to structural variants of a device, which helps to remedy the variant problem. More generally, it has been pointed out that the tasks which we want to accomplish involve changing or revising the behavior models themselves. In order to do this, it is first necessary to make explicit the possible modifications of models in terms of different behavior modes. Second, in order to do this efficiently, it is crucial to devise a conceptual modeling layer that allows to re-use unaffected parts of the model during revision. With this background, we can identify one main reason why the behavior models currently used in the automotive industry, e.g. for tasks such as simulation, can hardly be re-used for different purposes: they lack this conceptual modeling layer and do not make explicit possible model revisions. Consequently, it is hard to adapt these models to tasks other than the ones they have been originally designed for.

3.8 Summary

In this chapter, we have first analyzed how two basic problem solving tasks — behavior prediction and diagnosis — can be supported by behavior models. Next, we have introduced mathematical relations (i.e. constraints) as an universal means to formally describe behavior models. In this representation, elements of a relation (i.e. tuples) correspond to states of the system model.

We have derived necessary preconditions on the structure of relational behavior models. In particular, it is necessary to revise the model during tasks such as diagnosis. In order to accomplish this revision efficiently, the model is structured into two layers: a mathematical level (constraint network) and a conceptual layer. Model revision is done on the conceptual layer, which allows to identify and reuse parts of the mathematical level which are not affected during revision. The precondition is to organize a model into model fragments, which can be stored in a library.

We presented a commonly used ontology for this conceptual layer that consists of components and interface variables. This component-oriented ontology is suited for the application domain of automotive systems. We then defined how conceptual elements are mapped to a constraint network in the model composition step. Once a constraint network is obtained, various solution algorithms can be applied in order to solve it. This problem can also be cast as deriving inferences for a logical theory. The results can be interpreted as behavior prediction that determines the possible states of the system. Model-based diagnosis can be carried out in the case where the set of predicted states is empty, i.e. a fault has been detected. Different notions and algorithms have been presented that capture the tasks of fault localization and fault identification.

The outlined framework for model-based problem solving has a number of useful properties, such as completeness of the results. Given a behavior model of the device under consideration, we are thus equipped with methods to perform the tasks identified in the previous chapter.

However, we have not yet dealed with the problem of how to get such a model in the first place, in particular, if the physically possible behaviors of the device components are not completely known. The next chapter describes how this can be achieved.

Chapter 4

Qualitative Abstractions of Models

This section describes techniques how to deal with incompleteness of knowledge in the context of model-based problem solving. Incompleteness can refer to incomplete knowledge about the behavior of a device, incompleteness of the available input to the behavior model (e.g., observations), and incomplete specification of the aspects of the outcome that one is interested in. It is shown that in order to yield sound results, incompleteness has to be considered in all parts of the problem solving process, and not just in isolated aspects thereof (e.g., only during consistency checking).

Qualitative abstraction is the basic approach to handle this problem. It aims at including only the essential information in a behavior model. Fundamental properties of qualitative abstractions, in particular domain abstractions, are reviewed in this chapter. The second part of this chapter outlines the state of the art in deriving qualitative behavior models. It becomes apparent that in most approaches, a certain level of abstraction is implicitly assumed or pre-defined in the problem solving process. Moreover, it is argued that currently, there exists no first principles methodology to construct qualitative abstractions, starting from a ground representation and a characterization of the problem solving task.

4.1 Representing Physical Behavior

Chapter 3 provided us with a framework to define relational behavior models and to use them for fundamental problem solving tasks such as diagnosis and behavior prediction. As a first requirement imposed on the behavior models, we have seen that they have to be organized in such a way that they enable the steps of model composition and model revision. As an answer to this requirement, the basic concepts of compositional modeling have been presented in the last chapter.

The representation of a device in terms of relations allows to express behavior information about a device by restricting the set of possible tuples. A second requirement is that we want to make sure that the relation defines a behavior

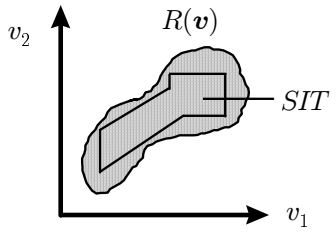

Figure 4.1: A relational behavior model covers the physically possible situations

model of the device, i.e. that it indeed describes (at least part of) the physical world. Therefore, we have to establish a link between the tuples in a relation and the possible situations that can occur for the device in the physical world. This is captured by the two definitions below. They follow the presentation given in [Str92b].

Definition 16 (Physical Situation) *Let D be a device. Let SIT be the set of situations which are physically possible for D. If, in a particular physical situation $s \in SIT$, $\boldsymbol{v}$ has the value $\boldsymbol{val} \in DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$, we write*

$$Val(s, \boldsymbol{v}, \boldsymbol{val}).$$

We will not further specify what physical situation are, nor how the set of physical situations can be characterized. We just use them in order to capture the relationship between a physical behavior and a model of this physical behavior. In particular, we can use this relationship to formally define a relational behavior model:

Definition 17 (Relational Behavior Model) *Let $DOM_1, DOM_2, \ldots, DOM_n$ be domains. A relation $R(\boldsymbol{v}) \subseteq DOM(v_1) \times DOM(v_2) \times \ldots \times DOM(v_n)$ specifies a behavior model of a device D, denoted $M(D, R)$, if and only if*

$$\forall s \in SIT : Val(s, \boldsymbol{v}, \boldsymbol{val}) \Rightarrow \boldsymbol{val} \in R.$$

The behavior model is called *strong* if the implication holds also for the reverse direction (see [Str92b]), and is called *weak* otherwise. The idea is that the tuples in a relational behavioral model of a device cover at least the set of behaviors that the device can exhibit. However, a behavior model that is weak might contain combinations of values that do not correspond to any physically possible situation. This is illustrated by Figure 4.1.

4.2 Incompleteness and Parsimony in Problem Solving

Incomplete characterizations of physical situations are ubiquitous, e.g. due to the granularity of observations, or due to the nature of available information about the behavior of system components (see chapter 2). On the other hand, the "true" physical behavior in a particular situation is singular, which implies that in any real-world application, behavior models are necessarily weak.

This aspect is referred to as *incompleteness* in model-based problem solving. It means that situations might be specified or be known only incompletely, thus always sets of physical situations rather than a singular, "true" physical situation have to be considered. To analyze this in more detail, section 4.2.1 identifies the possible sources of incompleteness in the model-based problem solving process.

Apart from incompleteness that is caused by the nature of available inputs to the problem solving process, it can also be useful to *purposefully* represent a device incompletely. The motivation is to have a parsimonious representation of a device that avoids any details in the model that would complicate the problem solving process itself, or that are unnecessary from the point of view of the result one is after. This is analyzed in more detail in section 4.2.2.

Section 4.2.3 is then concerned with requirements on the part of the behavior model in order to account for incompleteness and parsimony in the problem-solving process.

4.2.1 Dimensions of Incompleteness

Recall the formalization of the two basic problem-solving tasks — prediction and diagnosis — as given in chapter 3. The goal of prediction is to determine states (called SOL) under external restrictions called $CRITERIA$ that are consistent with the model:

$$MODEL \cup CRITERIA \vdash SOL.$$

The goal of diagnosis is to detect discrepancies of the model with external restrictions called OBS:

$$MODEL \cup OBS \vdash^{?} \bot.$$

Prediction and Diagnosis can be affected by incompleteness in a number of possible ways:

- For diagnosis, OBS can be incomplete. Possible reasons are, for instance, the imprecision of sensor signals of measured variables (e.g. which yield a set of values, like an interval, rather than a single value), or non-measurable external variables (e.g. the influence of road conditions on the behavior of an anti-lock braking system; see section 2.3.4).

- For prediction, $CRITERIA$ can be incomplete, for instance, reflecting an incomplete specification of a hypothetical situation (e.g. a braking distance that is assumed to be "longer than normal").

- $MODEL$ can be incomplete, reflecting partial knowledge about component behavior (e.g. in terms of parameters or functions; see section 2.3.3). For instance, for a behavior model that describes a leakage fault in a pipe, it might be only known that it has an opening with non-zero diameter. However, the precise parameters of the leak concerning its diameter, shape, etc. remain unknown.

4.2.2 Dimensions of Parsimony

In addition to the dimensions outlined in section 4.2.1, the necessity to deal with sets of situations can also be imposed by the goals of

- simplicity of the problem solving process (i.e. "$\vdash$"), e.g. to achieve tractability or improved efficiency of problem solving, or in order to keep the library of model fragments small,

- intuitiveness and naturalness of the outcome (i.e. SOL), e.g. to focus on the distinction between correct or faulty behaviors, or in order to provide meaningful explanations by classifying the space of resulting behaviors into meaningful regions.

The first point refers to the fact that in the context of *automated* model-based problem solving, state descriptions must be "usable" in the sense that basic problem-solving steps such as consistency checking and inference can be carried out efficiently. Hence, for instance, interval-based methods that aim at characterizing sets of numerical states by specifying real-valued intervals for system variables ([RR84]) are not a general solution. Although interval-based representations allow to cover sets of physical behaviors, they might fail to fulfill the tractability requirement. For instance, consistency might be impossible to decide for expressions involving intervals. This illustrates that an accurate representation of behavioral states might be inoperative because it renders basic problem solving steps too complex. Unnecessary complexity in the characterization of behavioral states should be avoided from this point of view, because larger sets of states tend to increase the complexity of solutions algorithms, and tend to decrease the possibilities of re-use during reasoning (e.g., when using an ATMS for keeping track of inferences). For instance, complexity analysis of constraint satisfaction algorithms has shown that the amount of computation required for behavior prediction grows exponentially with the number of variables in the model ([Kui86]).

The second point refers to the fact that it might be desirable to have incomplete descriptions in order to avoid any details in the model that are unnecessary from the point of view of the desired outcome. In general, a model is unnecessarily

detailed if it makes distinctions that are irrelevant to the result of the problem-solving process. For instance, for the task of diagnosis, distinctions between states should be avoided if they do not contribute to the detection of inconsistencies or to the discrimination of behavior modes. E.g. for the pedal position sensor example in section 1.1.1, it was only necessary to make distinctions for the voltage of the potentiometer that correspond to the switch-over point of the switch. This illustrates that the adequacy of a description can depend on the required granularity of the outcome.

4.2.3 Coping with Incompleteness and Parsimony

As the previous sections have shown, it is inevitable that sets of situations, rather than single situations, occur during problem solving. The basic problem with sets of situations is that we cannot restrict ourselves to considering just one or a subset of these situations. Instead, all possible states that correspond to physical situations have to be represented, in order to maintain the model property (definition 17). If this is not the case, it might occur that behaviors are wrongly refuted during problem solving.

For instance, if we would use a leakage model that chooses a specific parameter for the leakage opening, situations can occur where a leakage is physically present, but its behavior model is refuted, because it restricts the opening parameter to a value that does not correspond to the "real" physical situation. If we want to avoid this, we have to ensure that the sets of states cover all the possible leakage faults. Hence, a crucial requirement during modeling is to maintain the model property. This implies that numerical simulation models, capturing only one situation at a time by assigning single numerical values to variables, are in general inadequate for model-based problem solving. In almost all cases, the behavior of the device that is physically present would become inconsistent during problem solving, and we could only hope that the remaining behaviors (or diagnoses) are in some sense "close" to the true physical behavior.

To remedy this, a common approach in numerical modeling is to consider incompleteness during the consistency check. That is, the predicted and the measured values are considered inconsistent only beyond a certain (absolute or relative) tolerance threshold Δ:

$$v_i = val_1 \wedge v_i = val_2 \wedge |val_1 - val_2| > \Delta \vdash \bot.$$

This approach lies at the basis of the signal range check methods used in current on-board diagnosis and monitoring procedures (see section 2.2.2 in chapter 2). However, this approach considers incompleteness only in an isolated way, in this case, during the consistency check. Following the considerations above, this approach is of limited use for model-based applications, because the interaction of system variables — i.e. the device behavior itself — is not considered. The applied threshold is fixed and independent of the actual behavior. Hence, it would only be adequate for the special case where the variables affected by a fault

correspond directly to the observed variables. In the general case, however, it is impossible to determine the tolerance threshold Δ independently of the system.

This illustrates that the systematic basis for a general solution is to deal with incompleteness (i.e. sets of situations) *within* the behavior model and *during* the process of reasoning. The next section deals with such general approaches that can express incompleteness within a behavior description.

4.3 Model Abstraction

The last section has identified as a basic requirement for behavior models to ensure coverage of sets of physical situations. A representation that aims at representing whole sets or classes of behavioral states is called an *abstraction*. This is based on the view that there exists a fine-grained, e.g. real-valued, behavior model that accurately describes the physical behavior of a device, but it is, due to the reasons outlined above, not accessible or not applicable during problem solving. Abstraction corresponds to a mapping from an original representation to a new representation. In the context of abstraction, the original representation is often called the ground (or base) representation, and the abstracted representation is called the transformed representation. An abstraction is called *sound* if the transformed representation contains the base representation as an element:

Definition 18 (Sound Abstraction) *Let D_{ground} be a space of ground descriptions and $D_{abstract}$ be a space of abstract descriptions consisting of sets of ground descriptions. An abstraction $\tau : D_{ground} \rightarrow D_{abstract}$ is sound, if for each $d \in D_{ground}$, $d \in \tau(d)$.*

According to chapter 3, a behavior model consists of variables, domains and relations. Abstraction might affect each of these constituents. Thus, there are three basic types of possible abstractions: abstraction of variables, abstraction of domains, and abstraction of the relations between variables. Of course, combinations of these basic types are possible.

4.3.1 Domain (Value) Abstraction

First, we consider the case of model abstraction through the abstraction of domains. We pay special attention to this type of abstraction because it is probably the most obvious case, and it will be the starting point for the techniques described later in the thesis. As noted above, this type of abstraction can be expressed by a mapping of the elements of a ground domain to a transformed domain:

Definition 19 (Domain Mapping) *Let $DOM(v_i)$ be a domain. A domain mapping is a total mapping*

$$\tau_i : DOM(v_i) \rightarrow DOM'(v_i), val \mapsto val'$$

that maps elements of $DOM(v_i)$ to a domain $DOM'(v_i)$.

For a sound domain abstraction, $val \in val'$ and $DOM'(v_i) \subseteq 2^{DOM(v_i)}$. In a transformed model based on sound domain abstraction, the domain mapping τ_i aggregates the values of $DOM(v_i)$, i.e. the transformed domain is of the same size or of smaller size than the base domain. The aggregated values, i.e. the elements of the transformed domain $DOM'(v_i)$, are termed *qualitative values*:

Definition 20 (Qualitative Value) *A qualitative value $val' \in DOM'(v_i)$ denotes a (implicitly or explicitly specified) subset of values from the ground domain of a variable v_i:*

$$val' \in 2^{DOM(v_i)}.$$

For larger or infinite domains, it is infeasible or impossible to represent the ground values corresponding to a qualitative value by explictly enumerating them. Instead, they have to be described implicitly. Intervals are examples of implicitly specified qualitative values. If the ground domain is ordered, the ground values can be implicitly represented in a compact way by specifying lower and upper boundaries.

The boundaries can be thought of a representing the limits of regions where a change in behavior occurs, e.g. the freezing point or the boiling point of water. In the context of qualitative values, such boundaries are called *landmarks*. For instance, qualitative values for the ground domain of real numbers can be given by an ordered set of landmarks $l_1 < l_2 < \ldots < l_k$ and the open intervals around them:

$$(-\infty, l_1), l_1, (l_1, l_2), \ldots, l_k, (l_k, \infty).$$

This is the representation for qualitative values that underlies QSIM ([Kui86], [Kui94]; see also section 9.1). For a real-valued ground domain, the value zero is a prototypical landmark. It separates the set of positive values from the set of negative values. The transformed domain corresponding to this domain abstraction is called the sign domain:

Example 3 (Sign Domain) *Let $DOM(v_i)$ be the set of real values. The sign domain $DOM'(v_i)$ is defined by*

$$\tau_i(val) = \begin{cases} minus & : & val < 0 \\ zero & : & val = 0 \\ plus & : & val > 0 \end{cases}$$

□

The sign domain is widely used in model-based reasoning because it has a number of desirable properties ([Str90], [Wil92]; see also section 4.4.1).

Like in example 3, of special interest are qualitative values that consist of disjoint sets of ground domain values. Since domain mappings τ_i are total, non-overlapping qualitative values form a partition of the ground domain:

Definition 21 (Domain Partition) *A partition π_i of a domain $DOM(v_i)$ is set of non-empty disjoint subsets P_j of $DOM(v_i)$ that together cover the entire domain:*

1. $\forall j : P_j \in 2^{DOM(v_i)} \setminus \emptyset$ *(non-emptiness),*
2. $P_j \cap P_k \neq \emptyset \Rightarrow P_i = P_k$ *(mutually exclusiveness),*
3. $\bigcup_j P_j = DOM(v_i)$ *(exhaustiveness).*

There is a duality between domain mappings with disjoint qualitative values and partitions of domains: a domain abstraction could in this case either be understood as a mapping from a ground domain to an explictly defined new domain that consists of sets of elements of the ground domain, or as a partition of the ground domain where the partition elements implicitly define a new domain. Depending on the purpose, the first view or the second might be more convenient.

Proposition 1 (Duality of Domain Partitions and Domain Mappings) *A partition π of a domain DOM defines a domain mapping with disjoint qualitative values and vice versa.*

Because domain partitions are a central concept, we will define relationships between domain partitions. The definitions below capture the refinement and merge of two partitions:

Definition 22 (Refinement of Domain Partitions) *Let π_1, π_2 denote partitions of a domain DOM. π_1 is called a refinement of π_2, if*

$$\forall P_i \in \pi_1 : \exists P_j \in \pi_2 \text{ s.th. } P_i \subseteq P_j.$$

π_1 is called a strict refinement, if, additionally, $\pi_1 \neq \pi_2$.

If a partition π_1 is a (strict) refinement of π_2, we say conversely that π_2 is a (strict) *abstraction* of π_1.

Definition 23 (Merge of Domain Partitions) *Let π_1, π_2 denote partitions of a domain DOM. The merge of π_1, π_2 is the partition that contains all intersections of their elements:*

$$MERGE(\pi_1, \pi_2) := \{P_i \cap P_j | P_i \in \pi_1, P_j \in \pi_2\} \setminus \emptyset.$$

Domain mappings τ can be generalized straightforwardly from ground domain values to qualitative values $val' \in DOM'$ by defining

$$\tau(val') := \bigcup_{val \in val'} \tau(val).$$

Hence, a domain abstraction can itself be applied to qualitative values, and domain abstractions can be concatenated to form a hierarchy of abstractions. A set of domain mappings is denoted

$$\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_n).$$

The application of $\boldsymbol{\tau}$ to the tuples of a relation $R(\boldsymbol{v})$ yields the transformed relation

$$\boldsymbol{\tau}(R) := \{(\tau_1(val_1), \tau_2(val_2), \ldots, \tau_n(val_n)) \mid (val_1, val_2, \ldots, val_n) \in R\}.$$

We will also apply the basic operations on relations (section 3.2.2) to abstracted versions of relations. If an operator combines a relation on a set of ground domains with a transformed relation, we demand, for convenience, that the result is a relation defined on the ground domains. E.g., $R_1 \bowtie \boldsymbol{\tau}(R_2)$ will be used as a shorthand notation for the relation

$$R_1 \bowtie \{\boldsymbol{val} \mid \boldsymbol{val} \in val'_1 \times val'_2 \times \ldots \times val'_n, (val'_1, val'_2, \ldots, val'_n) \in \boldsymbol{\tau}(R_2)\}.$$

Because of the duality of partitions and domain abstractions, we will apply the notion of refinement, merge and concatenation both to domain partitions and domain abstractions.

Example 4 (Domain Mapping for Switch Model) *Let RC denote the behavior model of the component Switch as presented in example 1, together with the definition of its parameter $pos_{switching}$ as presented in example 2. Consider a domain partition for the domains of $v_{left}, v_{right}, v_{switch}$ that consists of three partition elements:*

$$\{\{[0V, 2V)\}, \{[2V, 4V), [4V, 6V), [6V, 8V)\}, \{[8V, 10V)\}\}.$$

Let $\boldsymbol{\tau}$ denote a domain abstraction for the variables in the switch model that corresponds to the above partition for variables $v_{left}, v_{right}, v_{switch}$, and is equal to the identical mapping for the remaining variables. Then the transformed relation $\boldsymbol{\tau}(RC)$ consists of only 54 *tuples, whereas the original relation RC consists of* 150 *tuples.*

□

4.3.2 Relation (Function) Abstraction

The second type of abstraction affects the relation types between the variables of a model. Relation abstraction maps relation types of a model to sets of relation types. The precondition is that the base relation types and the transformed relation types have the same scheme. For a sound relation abstraction, the tuples of the original relation type must be included in the transformed relation type. Most frequently in this approach, for simplicity only the special case of relation types that correspond to functions is considered.

Definition 24 (Qualitative Function) *Let S be a scheme. A qualitative function denotes a (implicitly or explicitly specified) subset of functions from the set of all functions over S.*

For instance, one might want to abstract the non-linear dependence of friction on the mass of a car into the class of monotonic functions. QSIM ([Kui86, Kui94]) provides such a representation for qualitative functions that correspond to the set of monotonically increasing or decreasing functions.

Since knowledge about the monotonicity of a function provides only a weak constraint, qualitative functions can be additionally constrained by specifying so-called *corresponding values* ([Kui94]). Corresponding values are tuples of landmark values that the variables in a qualitative function take on at the same time. Thus, they provide additional constraints on the sets of functions. For instance, a corresponding value $(0, 0)$ for a unary monotonic qualitative function constrains the functions to pass through the origin.

One problem with function abstraction is that since the number of possible functions is large, further restrictions are necessary in order to derive useful results. E.g. in QSIM, the set of functions is restricted to so-called *sensible* functions ([Kui94]).

An advantage of this type of abstraction is that compared to value abstraction, additional knowledge about the behavior of a device can be captured and exploited for reasoning. In particular, knowledge about the *deviation* Δv_i of a variable v_i can be propagated if it is known that the involved functions are monotonic. For instance, the real-valued constraint $v_1 = v_2 + v_3$ implies that

$$\Delta v_1 = \Delta v_2 + \Delta v_3.$$

This is the basis for so-called deviation or comparison models ([Wel88, dJvR99], see also the example in section 7.2.2) that aim at describing the behavior of a system relative to some reference behavior.

4.3.3 Variable Abstraction

The third type of abstraction affects the vector of variables in a model. Variable abstraction maps variables of the base model to new variables of the transformed model, which are possibly a combination of several variables of the base representation. Compared to the other types of abstraction, variable abstraction strongly affects the representation of the behavioral space of a model. Thus it is, in general, more difficult to handle from a theoretical point of view. Variable abstraction is often closely related to specific application domains. For instance, spatial aggregation is an application of variable abstraction where infinitely many spatially distributed variables are mapped to a finite set of qualitative regions ([BKZY96], [Lun96]).

As another example, fitting approximations ([Wel92]) for physical equations can be cast as (unsound) variable abstractions where variables are eliminated from a ground model by restricting them either to zero or one.

4.4 Problem Solving with Model Abstractions

We have identified principled possibilities for abstracting a relational behavior model. The question is what properties a transformed behavior model can have, relative to properties of the ground behavior model. Of particular interest is the model property defined in section 4.1 which demands that the actual behavior of a device is covered by the model.

4.4.1 Properties of Model Abstractions

Struss [Str90] has developed a framework for analyzing the relationship between solutions (i.e., sets of states) obtained with a ground model and the solutions obtained with an abstracted model in the context of interval-based domain abstractions. It extends a framework originally used by [Kui86]. A later paper ([Str92b]) investigates properties of relation abstractions in the context of model-based diagnosis. In the following, we review essential results of this analysis. First, we observe that a sound abstraction of a behavior model is also a behavior model of the device:

Proposition 2 (Sound Abstractions are Model-Preserving) *Let R be a model and D be a device such that $M(D, R)$. If τ is a sound abstraction, then $M(D, \tau(R))$.*

This is the basis for using sound abstractions of behavior models for model-based problem solving. Even in the presence of incomplete information, behaviors that do not appear as solutions can be safely refuted, because it is guaranteed that they do not correspond to any physically possible behaviors. On the other hand, solutions might occur that do not correspond to physically possible behaviors. Two other fundamental properties that are of interest when reasoning with abstractions are *completeness* and *stability*:

Definition 25 (Completeness and Stability) *An abstraction is*

- *complete, if each solution of the abstract description covers at least one of the ground solutions,*
- *stable, if the abstract solutions remain the same for transformations of ground descriptions that do not change the ground solutions.*

Proposition 3 (Incompleteness) *Abstract reasoning methods are, in general, not complete, i.e. there exist abstract solutions which have no ground counterpart (called "spurious" solutions).*

For the case of domain abstractions, the reason for incompleteness is the *selection problem* ([RR84, Str90, Kui94]). It denotes the fact that techniques for reasoning with abstractions cannot enforce coherent selection of elements from the base domain for multiple occurrences of the same variable in a qualitative representation.

Proposition 4 (Instability) *Abstract reasoning methods are, in general, not stable.*

More specifically, domain abstractions with a finite number of values do not preserve the associativity of arithmetic operators such as addition or multiplication, except for the domain partition corresponding to sign abstraction. This implies that abstractions are in general not stable with respect to transformations that change the order in which arithmetic operators are applied. The latter point is the foundation for approaches that transform behavior descriptions at the level of the ground representation before applying abstractions. For instance, the idea in [Wil92] is to perform a symbolic reformulation of arithmetic constraints such that the selection problem is diminished (see sections 4.5.5 and 9.1).

4.5 Qualitative Models as Parsimonious Abstractions

The requirements outlined in section 4.2 motivate to have descriptions of sets of states that cover all physically possible situations and that are as concise as possible with respect to the purpose of model-based problem solving. This lead to the development of *qualitative modeling* methods ([WdK90], [FS92]), which aim at covering classes of behaviors through computational methods, reflecting only the *essential* (i.e. the *significant*) distinctions. Qualitative modeling emerged primarily from common-sense reasoning about the physical world. Thus, it was originally termed *qualitative physics* or *naive physics* ([Hay90]).

Soundness demands that behavior models have to describe classes of behaviors that cover all physical behaviors of the device at hand. Parsimony of behavior models demands that behavior models are abstracted enough to include only information that is relevant to the context of the problem solving task.

Hence, qualitative modeling can be characterized as the problem of finding a level of abstraction that simplifies as much as possible, but does not oversimplify. Now that we have characterized the goal of qualitative modeling, the question is how we can obtain qualitative models in a defined or even automated way. In the following, we review various approaches that can be found in the literature.

4.5.1 Automated Modeling through Model Selection

One of the first approaches to automated modeling consists of selecting an adequate model from a set of candidate models. The principle of automated modeling through model selection is to have a fixed hierarchy of models to choose from. Each element of the hierarchy consists of a complete (i.e., already composed) model of the device that captures its behavior at a pre-defined level of abstraction. The model at each abstraction level must be provided by the user.

The basic idea is then to start the problem solving process with the simplest (i.e., most abstract) model, and to switch to a more complicated (i.e. more fine-grained) model only if necessary. Addanki et al. ([ACP91]) describe an instance of this method where the space of models is represented as a graph. The nodes

of the graph correspond to models, while the edges of the graph correspond to abstractions that are applied in going from one model to the other. Struss ([Str92a]) describes a similar approach that is tailored to the diagnostic problem solving process. These approaches have in common that the space of models has to be pre-defined and explicitly represented. The strategies enable to choose a suitable model among a set of behavior models, but do not provide a means to construct these models automatically. The adequacy of an abstraction level is defined implicitly by the control strategy that defines how to switch from one model to another and the given problem to be solved.

4.5.2 Automated Modeling through Model Composition

Automated modeling through model composition avoids the need of having to represent the space of possible models explicitly, and frees the user from the burden of defining the models a priori. Instead of selecting an adequate model of a complete device, it aims at composing it from a set of model fragments for each component type.

Because the search space for possible combinations of model fragments is much larger compared to the selection of an already composed model, automated generation of models through composition requires to have an explicit notion of the task the model is used for in order to guide the search. This task is expressed in the form of a user query that needs to be answered based on a structural description of the device and a library of model fragments.

Falkenhainer and Forbus ([FF92]) describe an instance of this approach, using an ATMS-based mechanism in order to identify classes of model fragments for a component that are deemed relevant to the user query. Rickel and Porter ([RP94]) as well as Iwasaki ([IL95]) use automated model composition to construct a model for answering questions concerning the qualitative behavior of a system over time. Nayak ([NJA92, Nay95]) describes the automated composition of a model in order to answer queries about the causal relationship of variables. Nayak also shows that the general task of constructing an adequate model through composition is intractable. However, if the sets of model fragments for a component are organized into classes, such that elements of different classes are mutually exclusive, it is shown that the search can be decomposed into independent subproblems and, hence, it becomes tractable.

Automated modeling through model composition requires to have a library of pre-defined classes of model fragments for the component types. Because it is difficult to specify model fragments a priori that will combine with each other in a coherent way, it is difficult to give guarantees on the outcome of the composition step. In particular, it can in general not be ensured that the composed model is a sound abstraction. Instead, heuristic strategies have to be used in order to choose suitable model fragments. For instance, it cannot be guaranteed that the device model resulting from the method described in [Nay95] indeed allows to derive the most parsimonious causal explanation of its behavior. Similarly, the approach described in [RP94] does not provide guarantees of soundness or

simplicity of the model. Another problem is that since the model is composed relative to a single query only, it will be rather specific and thus difficult to adapt to a modified task.

4.5.3 Automated Modeling through Model Transformation

Automated modeling through model transformation tries to avoid some of these problems by making the abstraction step more explicit. While automated modeling through model composition offers only two options for a model fragment (it can either be included in the behavior model or not), the idea of automated modeling through model transformation is to devise a set of abstraction operators that automatically transform model fragments (and thus models that have been composed thereof) to different levels of representation. The repeated application of the abstraction operators then yields a hierarchy of model abstractions. In general, the result depends on the specific sequence in which the operators are applied. Soundness of the abstraction can be ensured by carefully designing the abstraction operators. Since this can be difficult in general, the devised operators are often specific to a particular application domain.

For instance, Mauss ([Mau98]) and Ranon ([Ran98]) describe model transformation for a specific set of component types that occurs in the domain of electrical circuits. Struss and Heller ([SH98] describe the automated transformation of process-oriented models in the context of ecological domains.

Model transformation can provide a clearer semantics than model composition, and offers a still more flexible approach than the selection or composition of pre-defined models or fragments. However, the mentioned approaches lack a notion of adequacy of a level of abstraction. There is no criterion to decide when the application of the transformation operators should be stopped. Consequently, the model will be abstracted just as much as possible, i.e. as long as one of the operators is applicable. Transformation and composition can also be combined with each other in order to further transform a model that has been automatically composed. [Nay95] is an example of such a combined approach where first a model is composed that contains behavioral knowledge relevant for a query, and then becomes simplified by substituting model fragments with simpler ones, following a simplicity order that is predefined in the model fragment class.

4.5.4 Reasoning about Relevancy

A problem with the approaches described in the sections above is that the relevancy of features or distinctions within the model is built into the control sequence of the modeling system. This has motivated a number of approaches that aim at reasoning explicitly about relevancy and parsimony ([LIM92]).

To address the problem of deriving parsimonious descriptions that suppress unnecessary detail, Raiman [Rai91] describes an approach that is based on order-of-magnitude reasoning. The idea is to define an abstraction of domain values that allows to focus on significant magnitudes only. It consists of a set of local

rules that describe how orders of magnitude — and thus "significance" — propagates. This allows to draw conclusions like the following: imagine an object m_1 with a very large mass hitting an object m_2 with a very small mass head-on. If m_1 and m_2 had a similar velocity prior to the collision, then the velocity of the two objects afterwards will be similar to the initial velocity of object m_1. However, the rules described in [Rai91] lack a rigorous mathematical basis, in the sense that significance is defined only locally. As a consequence, the resulting abstraction is not sound. For instance, the rules do not cover the case where a large number of locally insignificant effects finally sum up to a significant effect. However, Raiman's approach can be seen as one of the earliest approaches to reason explicitly about relevancy in the context of physical systems. Nayak ([Nay95]) presents a variant of this approach that has a more formally defined semantics based on logarithmic scales.

Iwasaki ([Iwa92]) and Kuipers ([Kui94]) describe an approach for reasoning about relevancy in the context of temporal behaviors. This so-called time-scale abstraction is based on classifying dynamic behaviors into predefined categories such as instantaneous, fast, and slow, which can be associated with time scales of seconds, minutes, and days, respectively. The basic idea is that during reasoning at a particlar time scale, the behaviors at faster time scales can be modeled as instantaneous, while the behaviors at slower time scales can be neglected in the model. However, the scope of this a priori definition of relevance is limited. In general, whether or not a behavior should be modeled as instantaneous cannot be decided a priori, as it depends on the interactions occuring in the particular model, the observability of variables, and the reasoning goals in terms of the questions that need to be answered with the model. Iwasaki ([IFS+95]) refines the approach by linking the notion of instantaneous vs. non-instantaneous changes to measurability of durations, using a calculus of hyper-reals that is similar to order-of-magnitude reasoning.

4.5.5 Hybrid Algebras

Transformations of behavior models that do not affect the set of solutions on the ground level of representation can increase the set of solutions on the abstract level (see section 4.4.1). Thus, ad-hoc abstraction of mathematical expressions can lead to over-abstraction in the sense that applying suitable transformations to the expressions *before* abstracting them can yield a tighter set of solutions. Based on this observation, Williams ([Wil92]) defines a so-called hybrid algebra for automated abstraction. The idea of the system MINIMA, described in [Wil92], is to combine a symbolic (real-valued) representation of a behavior model with sign abstraction. Abstracting the domains of variables to signs simplifies the symbolic manipulation of mathematical expressions considerably. In MINIMA, shifting to the abstraction level of signs is postponed until the behavior model has undergone a sequence of symbolic manipulations that transform its equations to a unique normal form. The normal form eliminates the occurrence of the selection problem, hence MINIMA avoids problems caused by instability of abstractions.

Williams' approach thus captures the idea of obtaining — starting from a base model — optimal information with respect to a targeted level of distinctions, in this case, the signs of the variables. On the other hand, the approach is computationally demanding due to the involved symbolic reasoning. MINIMA is not concerned with the conceptual layer of models, and the specific domain abstraction to signs is hard-wired into the set of transformation rules defining the hybrid algebra.

4.6 Related Fields

In the following sections, we provide a survey of different approaches to abstraction from related fields, focusing particularly on domain abstraction.

4.6.1 Abstraction in Constraint Satisfaction

Abstraction, though a fundamental and powerful idea, has received only limited interest in the area of constraint satisfaction. The motivation to use abstraction in constraint satisfaction emerged primarily from its use to simplify and guide the solution process. The idea is that given a constraint satisfaction problem P to be solved, it can be transformed into a simpler problem P' such that P' can be more easily solved than P. Solving P' then can provide knowledge that makes the subsequent search for a solution to the original problem P easier. Techniques for transformation of constraint satisfaction problems either involve the abstraction of variables, the abstraction of values, or the abstraction of relations.

Interchangeability ([Fre91]) is an example for constraint abstraction based on the abstraction of domain values. It captures the idea of distinctions between values in a constraint satisfaction problem being redundant. Two values are interchangeable if exchanging one for another in any solution of a constraint satisfaction problem produces another solution. The definition below is adapted from [Fre91]:

Definition 26 (Full Interchangeability) *Let $R_1, R_2, \ldots R_m$ be a set of relations with variables $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ and solution $SOL(\boldsymbol{v}) = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$. Two values val_1, $val_2 \in DOM(v_i)$ are fully interchangeable iff every tuple of $SOL(\boldsymbol{v})$ which contains val_1 remains a solution when val_1 is substituted for val_2, and every tuple of $SOL(\boldsymbol{v})$ which contains val_2 remains a solution when val_2 is substituted for val_1.*

If two values val_1, val_2 are fully interchangeable, the only difference in the subsets of solutions involving val_1 and val_2 are val_1 and val_2 themselves. Therefore, sets of fully interchangeable values can be replaced by one meta-value, without losing any solutions. The sets of fully interchangeable values, i.e. the meta-values, form equivalence classes. Hence, it is only necessary to retain the meta-values in $DOM(v_i)$ instead of the values that constitute them. As a consequence, the domain size of the problem can be reduced. Replacing fully interchangeable values by a single new domain value corresponds to a special case of domain abstraction:

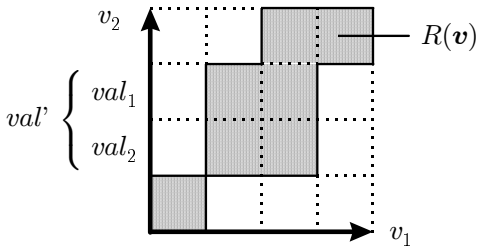

Figure 4.2: Domain abstraction corresponding to full interchangeability

Definition 27 (Interchangeability as Domain Abstraction) *For a relation $R(\boldsymbol{v})$, the domain abstraction $\boldsymbol{\tau_{FI,R}}$ corresponding to full interchangeability is defined by the domain partitions $\pi_i = \{P_{i,1}, P_{i,2}, \ldots, P_{i,k}\}$ given as*

$$val_1, val_2 \in P_{i,j} \Leftrightarrow \Pi_{\boldsymbol{v}\setminus\{v_i\}}(\sigma_{v_i=val_1}(R)) = \Pi_{\boldsymbol{v}\setminus\{v_i\}}(\sigma_{v_i=val_2}(R)).$$

In $\boldsymbol{\tau_{FI,R}}$, two values val_1, val_2 appear in the same partition element if and only if they belong to tuples of the relation that differ only in the value for variable v_i, and are equal for all the other variables. Graphically, this means that tuples involving val_1, val_2 form a rectangular "block". This is illustrated by figure 4.2.

[Fre91] also defines a local form of interchangeability, called neighborhood interchangeability, for binary constraints:

Definition 28 (Neighborhood Interchangeability) *Let $R_1, R_2, \ldots R_m$ be a set of binary relations with variables $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. Two values val_1, $val_2 \in DOM(v_i)$ are neighborhood interchangeable iff for every relation R_j involving v_i,*

$$\{val|(val_1, val) \in R_j\} = \{val|(val_2, val) \in R_j\}.$$

Neighborhood interchangeability is a sufficient, but not necessary, condition for full interchangeability. Sets of values that are neighborhood interchangeable can be determined in polynomial time. Neighborhood interchangeability and full interchangeability are special cases of k-interchangeability ([Fre91]), where $k - 1 \leq n$. With this generalized notion, 2-interchangeability is equivalent to neighborhood interchangeability, and n-interchangeability is equivalent to full interchangeability. [Fre91] also describes how interchangeability of values can be computed using a data structure called discrimination tree. [CN98] extends this procedure with the goal to localize independent subproblems in a constraint satisfaction problem. Choueiry and Noubir also show that for a special class of constraints expressing mutual exclusion between values, interchangeability corresponds to a decomposition of the original constraint satisfaction problem into locally independent subproblems. [FS95] uses neighborhood interchangeability in order to support abstraction and reformulation in the process of constraint satisfaction. [WF97] describes an iterative abstraction method for constraint satisfaction problems that is based on local interchangeability.

Conceptually, local forms of interchangeability can be viewed as complementing local constraint processing techniques, such as arc consistency ([Tsa93]), which aim at removing domain values that can not participate in any solution.

4.6.2 Abstraction in Theorem Proving

Theorem proving is concerned with the automation of proof construction in formal theories. The most common use of abstraction in theorem proving is to abstract the theorem to be proved (often referred to as the goal), and then to use the structure of a proof found for this abstracted goal to guide the construction of a proof for the original goal.

Giunchiglia and Walsh ([GW89]) propose a formalization of abstraction in the context of theorem proving. Abstractions are functions which map one representation of a formal system (called the ground representation) into a new representation (called the abstract representation). A formal system consists of a language, a set of axioms and a deductive machinery. Note that due to the duality outlined in section 3.2.4, problem solving using relational models can be understood as an instance of theorem proving, and model abstractions can be cast as abstractions of formal theories.

The goal is to find abstractions that preserve certain desirable properties of the original formal system, which leads to a classification of abstractions into two basic types. T^*-abstractions classify on whether or not provability is preserved between the ground representation and the abstract representation, whereas NT^*-abstractions classify on whether the detection of inconsistency is preserved. Of particular interest in our context are TI (theorem increasing) and NTI (non-theorem increasing) abstractions ([GW89], [GW92]):

Definition 29 (TI/NTI abstraction) *Let an abstraction f be a mapping from one formal system Σ_1 to another formal system Σ_2. Let $TH(\Sigma)$ denote the set of well-formed formulas that are theorems of a formal system Σ, and $NTH(\Sigma)$ denote the set well-formed formulas that are non-theorems of Σ. The abstraction f is TI, iff for any well-formed formula α, $\alpha \in TH(\Sigma_1) \Rightarrow f(\alpha) \in TH(\Sigma_2)$. The abstraction f is NTI, iff for any well-formed formula α, $\alpha \in NTH(\Sigma_1) \Rightarrow f(\alpha) \in NTH(\Sigma_2)$.*

An abstraction is TI iff for any theorem in the ground representation, there exists a corresponding abstract theorem. An abstraction is NTI iff for any well-formed formula that is not a theorem in the ground representation (i.e., that yields an inconsistency), its abstraction yields an inconsistency when added to the abstract representation of the formal system.

Domain abstractions are TI/NTI-abstractions in this terminology. Giunchiglia and Walsh show that a major problem with the use of TI/NTI-abstractions is that even if the ground formal system is consistent, the abstract formal system may be inconsistent. This problem is termed the "false proof problem" in [Pla81]. It is a major obstacle to the use of TI/NTI-abstractions to guide proofs in the ground representation, because if the abstract formal system is inconsistent, any

well-formed formula of this system is a theorem and thus it provides no information when used in filtering out non-theorems in the ground formal system. It turns out that any TI/NTI-abstraction is subject to the false proof problem, i.e. for any TI/NTI-abstraction, there exits a set of consistent axioms whose abstraction is inconsistent. Hence, adequate abstractions cannot be determined independently of the formal system at hand. This illustrates that the question of adequate abstractions cannot be answered satisfactorily from a pure logic point of view, as there are no universally valid criteria to prefer one abstraction over the other. In the context of model-based problem solving, this implies that an adequate abstraction depends both on the behavior model and the characteristics of the task it will be used for.

4.7 Discussion

Qualitative modeling provides a means to deal with incompleteness of knowledge in models and data, as present in real-world applications, and enables to achieve complete coverage of situations, as required by safety-critical applications such as FMEA or on-board diagnosis. Qualitative abstractions cover *classes* of systems and components. From a practical point of view, this helps to keep the model library small through avoiding irrelevant distinctions between component types.

More fundamentally, qualitative abstraction is the prerequisite for being able to reason about conceptual elements, such as the behavior modes of a component, explicitly. Without the step of abstraction, we could not talk about component *types* — e.g. for a resistor or a valve — to be revised or re-used in different contexts, but would instead be confronted with an infinitely large (and thus intractable) number of specific component instances.

Deriving qualitative abstractions is thus a highly practical requirement. It requires, in general, a notion of the *purpose* the model is used for. Usually, this purpose is understood as a user query about the interdependence of certain variables that needs to be answered based on a model. However, this notion of the purpose of model-based problem solving is of limited use, regarding the aim to support the principled tasks outlined in chapter 3. For instance, it would not be helpful in order to construct a model that is specifically suited for a diagnostic task where the goal is to discriminate among different behavior modes of a component, or for a prediction task where the goal is to decide whether a certain output variable, like the braking force, is above the required level. Achieving this requires a more general notion that takes into account more of the goals and conditions of problem solving. In particular, it is necessary to express what aspects of the outcome of the problem solving process are interesting or useful, and which inputs to the problem solving process (i.e., possible external restrictions) can occur or have to be considered.

The notion of a model purpose should be formal enough to serve as a sound criterion to drive the process of modeling. Existing approaches that incorporate a notion of model purpose (e.g. in the form of a user query) often cannot give guar-

antees that a certain transformation step makes the model more "optimal" with respect to the expressed purpose. For instance, in [Nay95], where the ultimate goal is to generate parsimonious causal explanations, it cannot be guaranteed that the transformed model indeed leads to a most parsimonious causal explanation. Instead, the transformations are only used as a heuristics in order to approach this goal.

Another point is that the transformation steps themselves must be sound in order to guarantee that the result of modeling is a sound abstraction. This was one of the difficulties involved in automated model composition. The model transformations described in [FF92] and [Nay95] do not correspond to any type of the abstractions defined in sections 4.3.1 to 4.3.3. The model simplifications devised in [Nay95] aim at including the relevant physical *phenomena*, and ensure that a causal explanation can be derived from the transformed model. While these aspects are important for the kind of task pursued in this work (generating causal explanations from the model), they are not directly related to the problem of achieving coverage of physical situations.

Approaches to reason about relevancy (section 4.5.4) aim at making the rationale behind transformation steps (e.g., identifying the gross behavior of a system) more explicitly in order to reason directly on this basis (e.g., small influences are subsumed by large influences). However, they often lack a sound mathematical basis. Since the number of rules that describe how relevancy propagates is necessarily limited, they can capture significance only through pre-defined, task-independent levels of abstraction.

Symbolic manipulations that transform the model before applying abstractions (section 4.5.5) can be designed to have defined mathematical properties, but they affect the structure of the model on the conceptual level. Applying such transformations can thus be problematic if the model later has to undergo model revisions, e.g. for the task of diagnosis.

4.7.1 Towards Qualitative Abstraction from First Principles

These considerations have motivated our goal to develop a first-principles-approach for deriving qualitative abstractions of behavior models from a ground representation, based on the following requirements:

(1) The applied transformation steps must be *sound abstractions* that belong to one of the abstraction types presented in sections 4.3.1 to 4.3.3. We will focus on domain abstraction (section 4.3.1) as the underlying abstraction type.

(2) There has to be a notion of *modeling goals* that allows to express what aspects of the outcome of the problem solving process we are after.

(3) There has to be a notion of *modeling conditions* that allows to express what inputs to the problem solving process (i.e., external restrictions) can occur.

(4) We need a criterion for *maximality* of an abstraction level, based on the condition that further abstraction steps applied to a maximally abstracted model would cause over-simplifications that would prevent one from reaching at least one of the modeling goals.

(5) The method should be applicable to *arbitrary relational models*, and not be limited to restricted cases such as e.g. monotonic functions.

Hence, in addition to the ideas and concepts presented in chapter 3, our goal (and contribution) is to make explicit *task-dependency* in order to reason directly about such aspects as observability, desired distinctions, possible distinctions, necessary distinctions, and unnecessary distinctions in a model. The ability to explicitly reason about task-dependency is the prerequisite to automate the task of finding the right granularity of a model. The focus is on domain abstraction (sections 4.3.1), i.e. the goal is to find — based on the above requirements — suitable qualitative values for the domains of variables.

4.8 Summary

In this chapter, we have been concerned with the impact of incomplete knowledge on the model, and techniques how to handle this in the models and the model-based problem solving process. In doing so, we have identified another limitation of existing problem solving techniques for automotive systems. Existing methods consider imprecision only during isolated parts of the problem solving process, such as the consistency check. However, in general, the relation between a fault and deviations of system variables cannot be determined a priori and locally, but must be derived from the system model. Abstraction is the general method to solve this problem. We have analyzed several types of abstractions, and focused particularly on domain abstraction. Fundamental properties of abstractions are soundness, incompleteness and instability. Besides the ability to deal with incomplete inputs to problem solving, abstraction is also the basis for purposefully representing a system incompletely, with the intention of avoiding any details in the model that are unnecessary for the result. Qualitative modeling is the task of finding a level of abstraction that includes only the relevant distinctions in the model. Automatically deriving qualitative abstractions is of highly practical importance in order to make the idea of model-based systems feasible. Methods for automated modeling have been presented that are based on selecting, composing or transforming a model with an adequate level of abstraction. Often, such methods are based on predefined abstractions only, lack an explicit or general notion of the purpose the model is used for, or cannot guarantee that the resulting model is sound or adequate with respect to the specified purpose. This is the motivation for developing a method that allows to automatically derive qualitative abstractions of relational models, based on a sound type of abstractions and an explicit notion of desired outcomes and available inputs of the problem solving process. This will be the topic of the next chapter.

Chapter 5

Task-dependent Qualitative Model Abstraction

In this chapter, we develop the necessary apparatus to formally characterize our goal, which was up to now informally described as having the "right model for the particular task". To this end, we have to define more precisely what is meant by "model", by "task", and by "right", respectively.

The model is given by a compositional, relational behavior model as described in chapter 3. A particular task is characterized by the purpose to obtain solutions of model-based problem-solving tasks at a specified level of granularity, based on external restrictions that are available at a particular level of granularity. These two aspects of task-dependency are captured by *target distinctions* and *observable distinctions*, respectively.

What is "right" is then defined as the requirement to have a model that is maximally abstracted, but still allows to achieve the purpose equally well as the base model. We restrict ourselves to the abstraction type of domain abstractions (section 4.3.1). Observable distinctions that are coarser than the base model can give rise to abstractions in a domain as certain distinctions *cannot* be made. Conversely, target distinctions that are coarser than the base model can give rise to abstractions in a domain because certain distinctions *need not* be made.

A qualitative abstraction problem can then be captured as a triple that consists of a relational model and two domain abstractions that specify observable distinctions and target distinctions. Solving a qualitative abstraction problem means to incorporate only distinctions in a domain that are both necessary and sufficient for the purpose, i.e. finding a set of domain abstractions that is both *distinguishing* and *maximal*. This formalization allows to investigate fundamental properties related to the existence of solutions and the computational complexity of the problem of determining qualitative values for the domains of variables. These results lay the basis for the computational solution presented in the next chapter.

5.1 Task-dependency in Problem Solving

In chapter 3, we have defined a model-based reasoning framework that consists of a space for defining behavior models, a space of observations (i.e. external restrictions), and a space of solutions. The basic tasks of behavior prediction and diagnosis can be formalized in terms of these concepts. Figure 5.1 summarizes this framework graphically.

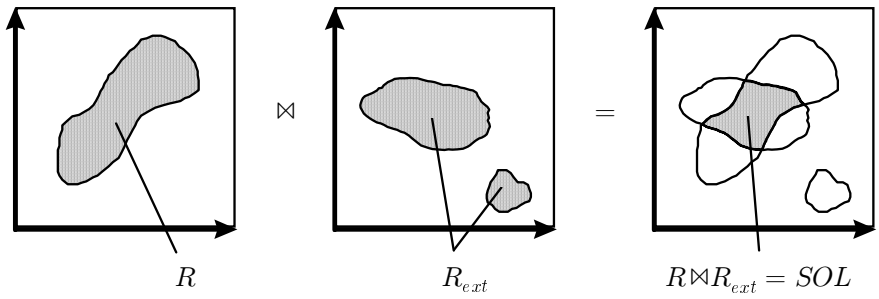

Figure 5.1: Framework for model-based problem solving: A model relation R together with external restrictions R_{ext} yields solutions SOL

Chapter 4 described how incompleteness of behavioral knowledge can in principle be accounted for through abstractions. Qualitative models have a level of abstraction that is appropriate in the sense that it includes only the distinctions that are essential to derive the solutions.

Our ultimate goal is to derive such qualitative abstractions. Thus, we first need criteria that allow us to decide whether a certain level of abstraction is appropriate for a certain task or not. The appropriateness of a model depends on *task-dependent* characteristics, which are given by

- the *context* the model is used in, as determined by the set of possible external *restrictions* (e.g., observations corresponding to measurements),
- the *purpose* the model is used for, as determined by the set of possible *solutions* that we want to discriminate (e.g., different behavior modes for diagnosis).

The set of external restrictions characterizes the available "inputs" to the model. The set of possible solutions characterizes the "outputs" that we are interested in when solving problems using the model.

Chapter 4 showed that both of these task-dependent characteristics can influence the appropriate granularity of the behavior model: if either the possible inputs or the possible results have a granularity that differs from the granularity of the base model, this can give rise to possible abstractions of the behavior model. The following sections address the problem of characterizing the adequate

granularity of behavior models relative to these characteristics of a task. In order to accomplish this, we first have to enhance our model-based problem solving framework to explicitly incorporate such task-dependent characteristics. This is described in the next section.

5.2 Task-dependent Distinctions

Within the framework outlined in the previous section, task-dependent characteristics can be captured by means of two different kinds of abstractions:

(1) the identification of states in the observation space that cannot be distinguished, given the granularity of external restrictions.

(2) the identification of states in the solution space that need not be distinguished, given the granularity of solutions.

The first type of abstraction, which identifies states that cannot be distinguished from each other, will be termed *observable distinction*. This name reflects the idea that distinctions expressed by observable distinctions constitute the granularity of the possible external restrictions. However, note that observations are only a special case of external restrictions, which could in general correspond to measurements, specifications given by the user, or hypothetical situations as considered e.g. in an FMEA.

The second kind of abstraction, which identifies solutions that need not be distinguished from each other, will be termed *target distinction*. This name reflects the idea that distinctions expressed by target distinctions constitute the possible solutions, and, hence, the "purpose" of abstraction.

We will capture task-dependent abstractions in the context of domain abstraction only. We do not consider relation abstraction (section 4.3.2) or variable abstraction (section 4.3.3). One reason is that the model-based reasoning framework that will be the basis for the implementation (see chapter 7) can, at the moment, only deal with value abstraction.

Hence, we are interested in abstractions of the state space that arise from domain abstractions. An *orthogonal partition* is a partition of the state space that consists of independent domain partitions for the individual variables:

Definition 30 (Orthogonal Partition) *Let* $\boldsymbol{v} = (v_1, ..., v_n)$ *be model variables,* $DOM(\boldsymbol{v}) = DOM(v_1) \times \ldots \times DOM(v_n)$*. An orthogonal partition is a partition of* $DOM(\boldsymbol{v})$ *that is defined by domain partitions for the individual* $DOM(v_i)$*, i.e.*

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n).$$

It has been shown in section 4.3.1 that a domain partition corresponds to a domain abstraction. Thus, $\boldsymbol{\pi}$ corresponds to a set of domain abstractions $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_n)$. We will capture the two types of task-dependent abstractions (target distinctions and observable distinctions) as orthogonal partitions. This is described in more detail in the next two sections.

5.2.1 Observable Distinctions

Observable distinctions are domain abstractions that identify states which *cannot* be distinguished from each other. They give rise to abstractions of a model because they introduce a "don't know" indeterminism among the behavioral states of a device.

Observable distinctions are abstractions reflecting the granularity of the inputs to the model, which might be coarser than the granularity of the base model e.g. due to incomplete observability. They are a means to express measurement granularity or even non-observability of intermediate variables. The latter case occurs e.g. in on-board diagnosis, where only certain variables that correspond to the sensor inputs are observable. Hence, conceptually, observable distinctions aim at capturing the kind of incompleteness that was considered in section 4.2.1 of chapter 4.

As noted in the previous section, an observable distinction is represented as an orthogonal partition of the observation space:

Definition 31 (Observable Distinction) *An observable distinction, denoted* $\boldsymbol{\pi_{obs}}$*, is an orthogonal partition of the space* $DOM(\boldsymbol{v})$*.*

Figure 5.2 illustrates observable distinctions graphically. A variable v_i is not observable at all if the domain partition for v_i specified by $\boldsymbol{\pi_{obs}}$ is equal to the trivial domain partition, i.e.

$$\pi_{obs,i} = \{DOM(v_i)\}.$$

Example 5 (Observable Distinction for Pedal Position Sensor) *For the pedal position sensor presented in section 1.1.1, the electronic control unit senses the output voltages of the potentiometer and the switch component, but cannot measure the other variables. This can be stated as*

$$\pi_{obs,v_{pot}} = \{\{[0V, 2V)\}, \{[2V, 4V)\}, \ldots, \{[8V, 10V)\}\},$$
$$\pi_{obs,v_{switch}} = \{\{[0V, 2V)\}, \{[2V, 4V)\}, \ldots, \{[8V, 10V)\}\}.$$

The other variables receive the trivial partition

$$\pi_{obs,i} = \{DOM(v_i)\}.$$

□

5.2.2 Target Distinctions

Target distinctions identify solutions that *need not* be distinguished from each other. They give rise to abstractions of a device model because they introduce a "don't care" indeterminism among its behavioral states.

For instance, in behavior analysis for FMEA, we might be interested in the values of certain output variables only, such as the torque of the engine or the

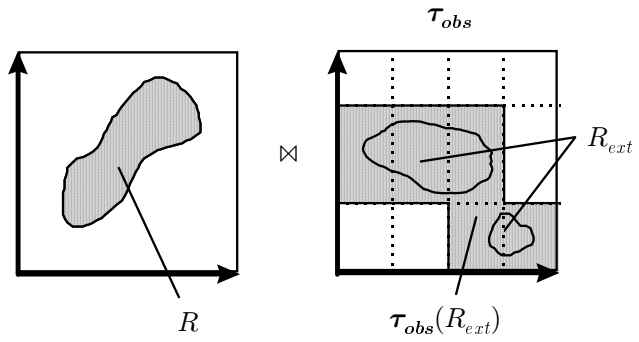

Figure 5.2: Observable distinctions define the granularity of external restrictions

braking force at the wheels. Values for intermediate variables such as the pressure at various points in the system or the flow into the brake cylinders are not interesting by themselves, but only useful if necessary to derive these results. As another example, consider the task of diagnosis, where we are interested only in the possible behavior modes of the components. For on-board diagnosis, it might even not be necessary to know the particular behavior mode of the components, but it is instead only necessary to distinguish such classes of behavior modes that require different recovery actions.

Conceptually, target distinctions are a means to express the kind of parsimony that was considered in section 4.2.2 of chapter 4. Analogously to observable distinctions, target distinctions are captured as an orthogonal partition of the solution space:

Definition 32 (Target Distinction) *A target distinction, denoted $\boldsymbol{\pi_{targ}}$, is an orthogonal partition of the space $DOM(\boldsymbol{v})$.*

Figure 5.3 illustrates target distinctions graphically. A variable v_i is said to have no target partition if the domain partition $\pi_{targ,i}$ is equal to the trivial domain partition $\{DOM(v_i)\}$.

Example 6 (Target Distinction for Pedal Position Sensor) *An example for a target distinction occurs for the pedal position sensor example presented in section 1.1.1. The initial goal to distinguish the voltage values ground $= [0V, 2V)$ and battery $= [8V, 10V)$ for the variable v_{switch} can be expressed (if the plausibility check itself is not represented in the model) as a target partition for this variable that separates $[0V, 2V)$ and $[8V, 10V)$ from the rest of the domain values:*

$$\pi_{targ,v_{switch}} = \{\{[0V, 2V)\}, \{[2V, 4V), [4V, 6V), [6V, 10V)\}, \{[8V, 10V)\}\}.$$

The other variables receive the trivial partition

$$\pi_{targ,i} = \{DOM(v_i)\}.$$

□

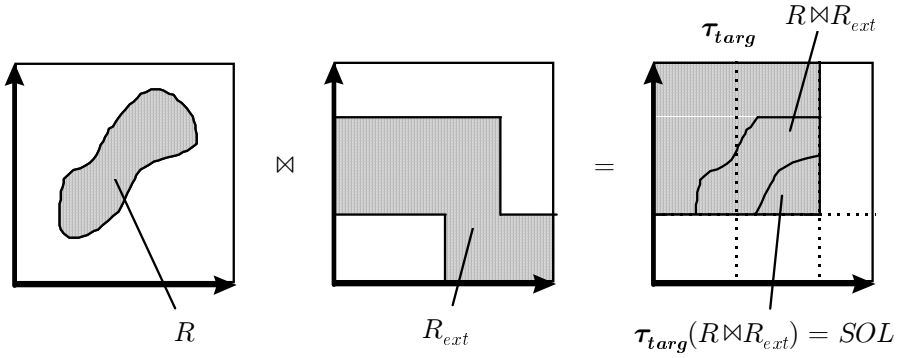

Figure 5.3: Target distinctions define the granularity of solutions

We have now augmented the framework for model-based problem solving by observable distinctions and target distinctions as a means to represent task-dependent characteristics explicitly. The framework is summarized in figure 5.4. A behavior model $R(\boldsymbol{v})$, combined with external restrictions R_{ext} of a given granularity $\boldsymbol{\tau_{obs}}$, leads to solutions that are distinguished at a granularity $\boldsymbol{\tau_{targ}}$ only.

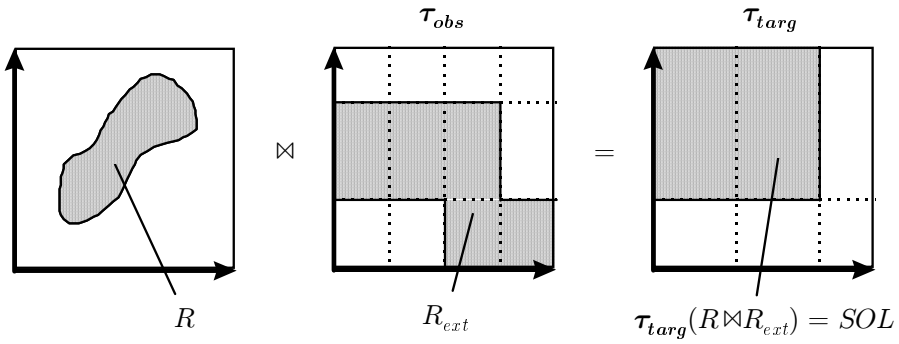

Figure 5.4: Framework for model-based problem solving using task-dependent distinctions

Equipped with the means to specify the conditions of a particular task, we now return to the problem of deriving abstractions of behavior models that are suited for this particular task. This problem is addressed in the next section.

5.3 Task-dependent Qualitative Abstraction Problem

Using the terminology introduced in the previous section, the problem of finding a qualitative model abstraction can be captured as follows. If we are given a behavior model R, a domain abstraction $\boldsymbol{\tau_{obs}}$ corresponding to observable dis-

tinctions and a domain abstraction $\boldsymbol{\tau_{targ}}$ corresponding to target distinctions, we would like to find a domain abstraction of the model that yields the same results as the original model. In other words, we would like to find an abstraction that incorporates all necessary distinctions, given the target and observable distinctions.

This guarantees that if we apply the abstraction to the behavior model, then for the considered external restrictions, the abstracted model will yield the same results as the original model. As a consequence, we can substitute the abstracted model for the original model in problem solving. A qualitative abstraction problem thus consists of the following ingredients:

Definition 33 (Qualitative Abstraction Problem) *A qualitative abstraction problem QAP is a tuple* $(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$*, where* R *is a relational behavior model,* $\boldsymbol{\tau_{obs}}$ *is a domain abstraction defined by observable distinctions, and* $\boldsymbol{\tau_{targ}}$ *is a domain abstraction defined by target distinctions.*

Of course, it is possible that even the base behavior model R itself cannot determine all distinctions defined by $\boldsymbol{\tau_{targ}}$. But with respect to the possible ones, the domains $DOM(v_i)$ of the base model may be overly detailed.

Solving a qualitative abstraction problem corresponds to determining distinctions to be made in the domains $DOM(v_i)$ that are necessary in order to reproduce their distinguishing power. This means finding a set of domain partitions

$$\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$$

which is able to preserve as much information as possible with respect to the target distinctions. The approach taken is based on the view that when using the model, it is sufficient to get optimal information about the resulting solutions, and that distinctions in the domain of a behavior model should be considered only if they are necessary to derive conclusions about these solutions.

5.3.1 Induced Distinctions

As stated above, the solutions to a qualitative abstraction problem are given by a set of domain partitions $\boldsymbol{\pi}$, which corresponds to a set of domain abstractions $\boldsymbol{\tau}$. If $\boldsymbol{\tau}$ contains sufficient distinctions to derive all information about the resulting solutions, it will be denoted *distinguishing domain abstraction*:

Definition 34 (Distinguishing Domain Abstraction) *Let* $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ *be a qualitative abstraction problem. A distinguishing domain abstraction for QAP is a set of domain abstractions*

$$\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_n)$$

such that for the considered external restrictions $R_{ext} \subseteq \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$*, the abstracted model derives a solution* $SOL \subseteq \boldsymbol{\tau_{targ}}(DOM(\boldsymbol{v}))$ *if and only if the base model derives the same solution:*

$$\boldsymbol{\tau_{targ}}(R \bowtie R_{ext}) = SOL \Leftrightarrow \boldsymbol{\tau_{targ}}(\boldsymbol{\tau}(R) \bowtie \boldsymbol{\tau}(R_{ext})) = SOL.$$

The requirement expressed in definition 34 means that if we are given an external restriction on the level of observable distinctions — actual observations, design specifications, etc. —, applying the distinguishing domain abstraction $\boldsymbol{\tau}$ *before* determining the result does not change the solutions on the level of the target distinction. Put in other words, the abstracted behavior model contains *sufficient* distinctions.

There might exist more than one possible domain abstraction that fulfills this criterion. In particular, the domain abstraction that corresponds to the identical mapping

$$\boldsymbol{\tau_{id}} : DOM(\boldsymbol{v}) \rightarrow DOM(\boldsymbol{v}), \boldsymbol{val} \mapsto \boldsymbol{val}$$

retains all the distinctions of the domains, and thus it is a distinguishing domain abstraction. Also, the domain abstraction corresponding to the merge of the observable and target distinctions

$$\boldsymbol{\tau_{merge}} := MERGE(\boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$$

contains sufficient distinctions to be a distinguishing domain abstraction. Hence, we also have to state the requirement that a qualitative abstraction contains only *necessary* distinctions. This means that in the case where there exists more than one distinguishing domain abstraction, we would like to find a maximal one. A maximal abstraction guarantees that any finer abstraction incorporates distinctions that are unnecessary, given the target and observable distinctions:

Definition 35 (Maximal Distinguishing Domain Abstraction) *Let QAP* $= (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ *be a qualitative abstraction problem. A distinguishing domain abstraction* $\boldsymbol{\tau_{ind}}$ *is a maximal distinguishing domain abstraction for QAP, if there does not exist a distinguishing domain abstraction* $\boldsymbol{\tau'_{ind}}$ *for QAP such that* $\boldsymbol{\tau_{ind}}$ *is a strict refinement of* $\boldsymbol{\tau'_{ind}}$.

A maximal distinguishing domain abstraction incorporates only distinctions that together are both necessary and sufficient according to the target and observable distinctions. It represents a level of abstraction that is most adequate to solve the problem, as it neither makes any unnecessary distinctions, nor does it abstract away any distinctions that are crucial to solve the problem. A maximal distinguishing domain abstraction thus captures the intuition behind a qualitative model. Definition 35 might be thought of as a formal definition of qualitative abstraction in the sense of section 4.5. More precisely, it formalizes the problem of finding qualitative values for the domains of variables: Since a maximal distinguishing domain abstraction $\boldsymbol{\tau_{ind}}$ corresponds to a set of domain partitions, finding $\boldsymbol{\tau_{ind}}$ means finding maximal partitions of the individual domains $DOM(v_i)$. The maximal partitions of the individual $DOM(v_i)$ then constitute sets of qualitative values for the individual variables v_i. However, since this holds only under

certain preconditions, e.g. the restriction to value abstraction, we prefer to use the term *induced distinction* instead of *qualitative abstraction* to avoid confusion with the general problem outlined in chapter 4:

Definition 36 (Induced Distinction) *An induced distinction, denoted $\boldsymbol{\pi_{ind}}$, is the partition of $DOM(\boldsymbol{v})$ corresponding to a maximal distinguishing domain abstraction $\boldsymbol{\tau_{ind}}$.*

An induced distinction expresses and formalizes our initial goal of determining qualitative values for the domains of the model variables from first principles.

Example 7 (PPS Switch) *Consider a subset of the pedal position sensor model described in example 1 that consists of the components Switch, Battery, $Node_1$ and $Node_2$. Let $\boldsymbol{\tau_{targ}}$ be given as in example 6, and let $\boldsymbol{\tau_{obs}}$ be equal to the identical domain mapping. Then the induced distinction for the switch state is*

$$\pi_{ind,Switch.state} = \{\{left\}, \{right\}\}.$$

Next, consider a structural modification where component Switch is connected to Battery as shown in figure 5.5. Then an induced distinction does not have to distinguish between the two switch states, because both states will yield the same result on the level of target distinctions:

$$\pi_{ind,Switch.state} = \{\{left, right\}\}.$$

This example illustrates the influence of the structural part of the model on the granularity of the induced distinctions.

□

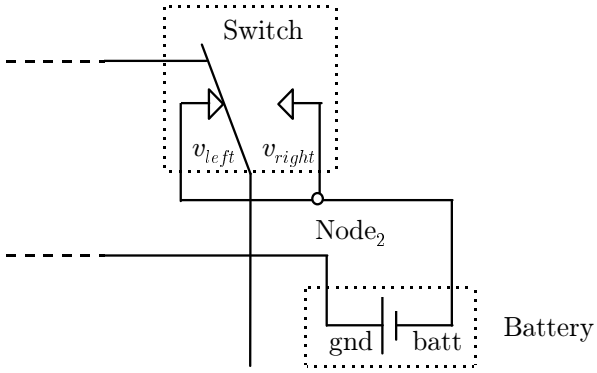

Figure 5.5: Switch with modified structure

Example 8 (Multiplication Constraint) *Let* $\boldsymbol{v} = (v_1, v_2, v_3)$. *Let* $DOM(v_i)$ *be equal to the real numbers,* $i = 1, 2, 3$. *Let* R *express the behavior* $v_3 = v_1 \cdot v_2$. *Assume that the only non-trivial target partition is given for* v_3*:*

$$\pi_{targ,3} = \{(-\infty, 0), 0, (0, 1), 1, (1, \infty)\}.$$

Let these partition elements be denoted $val_1, val_2, \ldots, val_5$. *Assume, first, that the observable partition for* v_1 *and* v_2 *is given by* π_{id}. *Then the induced distinctions for* v_1 *and* v_2 *are also equal to* π_{id}*:*

$$\pi_{ind,1} = \pi_{ind,2} = \pi_{id}.$$

To see this, consider an abstraction $\boldsymbol{\tau}$ *that maps two different reals* $a_1 \neq a_2$ *onto the same domain value. Then choosing the external restriction*

$$R_{ext} = \{(a_1, \frac{1}{a_1})\}$$

reveals the loss: e.g. for the case $0 < a_1 < a_2$, *the base relation yields the solution* $SOL = \{((-\infty, \infty), (-\infty, \infty), val_4)\}$, *whereas the abstraction* $\boldsymbol{\tau}(R)$ *yields* $SOL' = SOL \cup \{((-\infty, \infty), (-\infty, \infty), val_5)\}$ *(the other cases are similar). Now assume that the observable partition for* v_1 *and* v_2 *is given by a partition that consists of the integer values and open intervals between them:*

$$\pi_{obs,1} = \pi_{obs,2} = \{\ldots, -1, (-1, 0), 0, (0, 1), 1, \ldots\}$$

In this case, the picture changes. As suggested by figure 5.6, all values of v_1 *greater than 1 can be summarized. Intuitively, it would not pay off to distinguish between them because the values of* v_2 *are not fine-grained enough to exploit the distinction e.g. for determining whether* v_3 *is less than, equal to, or greater than 1. Therefore, the induced distinctions for* v_1 *and* v_2 *are given by*

$$\pi_{ind,1} = \pi_{ind,2} = \{(-\infty, -1), -1, (-1, 0), 0, (0, 1), (1, \infty)\}.$$

This example illustrates the influence of the granularity of the external restrictions on the level of abstraction that can be achieved by induced distinctions. □

The examples and definitions above raise a couple of interesting questions:

(1) In the case of finite domains, how many induced distinctions can occur in principle? I.e. what is the size of the search space for solutions?

(2) Under what conditions is it possible to derive induced distinctions? I.e. when can we guarantee that a unique solution does exist?

(3) What degree of abstraction can be achieved, for different combinations of problem solving tasks and given initial conditions? I.e. what determines the characteristics of a solution?

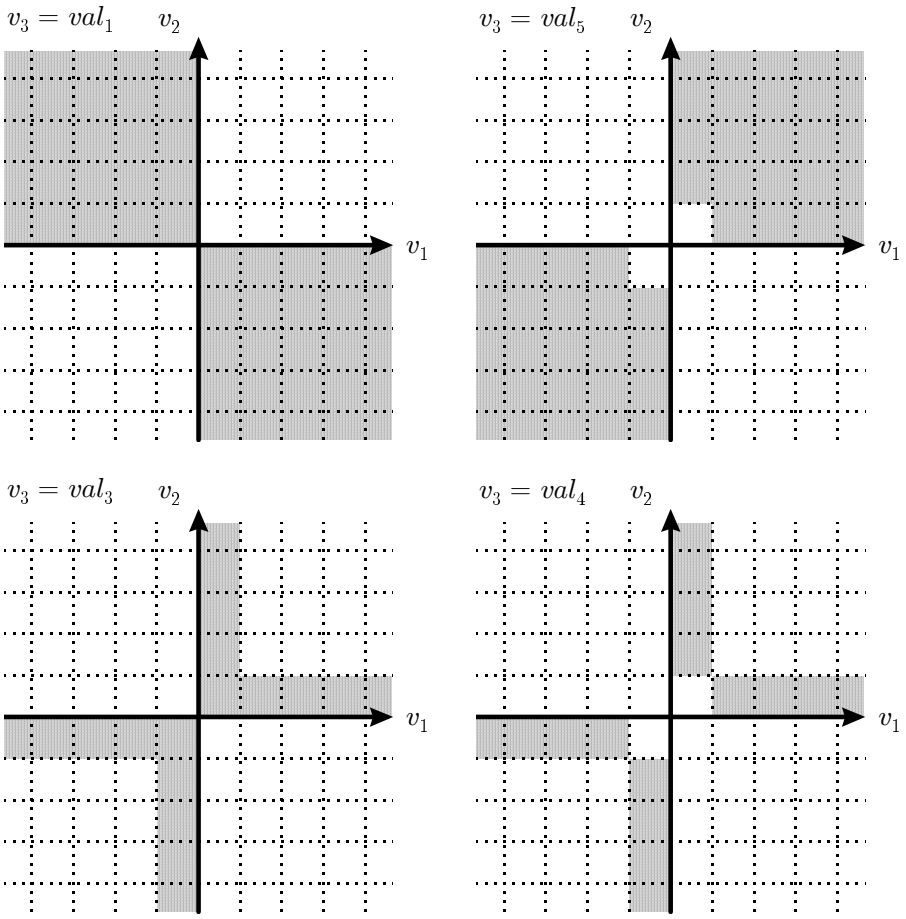

Figure 5.6: Projections of the multiplication constraint in example 8 on the qualitative values of v_3. The diagram for $v_3 = val_2$ (not shown) coincides with the axes v_1,v_2.

(4) How costly is it do derive induced distinctions? I.e. what is the complexity of deriving a solution computationally?

(5) Finally, what is the impact of the specific structure of models — i.e. compositional behavior models — that we want to use in the context of our application? I.e. can we exploit the ontological model layer during the process of deriving a solution?

Each of these questions will be addressed in this thesis. In the following sections, we investigate fundamental properties of induced distinctions concerning the size of the search space for solutions, the complexity of finding a solution, and the existence of unique solutions. This provides answers to the first four issues that were raised above. We also shed more light on the relationship of qualitative abstractions and concepts that have been developed in the field of

constraint satisfaction, in particular, interchangeability. This will provide an answer to the question how the characterization of $\boldsymbol{\tau_{ind}}$ can be turned to a form that is amenable for algorithmic computation. Thus, the results derived in the following sections will serve as the basis for devising the computational solution to task-dependent qualitative model abstraction that is presented in the next chapter.

5.4 Task-dependent Abstraction as a Search Problem

The last sections provided us with a formal definition of task-dependent qualitative model abstraction. The solution to this problem has been captured by induced distinctions that can be represented as a domain mapping $\boldsymbol{\tau_{ind}}$. In this section, we characterize the search space for induced distinctions for the special case of finite domains. If the domains of the variables in the model are finite, there is only a finite number of possible domain abstractions, and consequently, the problem of finding induced distinctions can be cast as a finite search problem.

5.4.1 Search Space for Induced Distinctions

The elements of the search space for a finite qualitative abstraction problem are given by the possible sets of domain abstractions $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_n).$$

A domain abstraction τ_i corresponds to a domain partition π_i. Hence, in order to determine the size of the search space, we have to analyze the number of possible partitions of a domain $DOM(v_i)$. For example, the domain

$$\{1, 2, 3\}$$

can be partitioned in the following five ways:

$$\begin{gathered}\{\{1\},\{2\},\{3\}\},\\ \{\{1\},\{2,3\}\},\\ \{\{2\},\{1,3\}\},\\ \{\{3\},\{1,2\}\},\\ \{\{1,2,3\}\}.\end{gathered}$$

The number of ways a set of n elements can be partitioned into k disjoint, non-empty subsets is called the Stirling number of the second kind ([And76]). It is given by

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i\binom{k}{i}(k-i)^n,$$

where $\binom{n}{k}$ denotes the binomial coefficient. The Stirling numbers can also be generated recursively using the formula

$$S(n,k) = S(n-1,k-1) + kS(n-1,k).$$

The number of ways a set of n elements can be partitioned is then equal to the sum of the Stirling numbers for $k = 1, 2, \ldots, n$:

$$B_n = \sum_{k=1}^{n} S(n,k).$$

The B_n are known as the Bell numbers ([And76]). Bell numbers can be computed by the recurrence scheme

$$B_{n+1} = \sum_{k=0}^{n} B_k \binom{n}{k}.$$

or using the formula

$$B_n = \left\lceil e^{-1} \sum_{m=1}^{2n} \frac{m^n}{m!} \right\rceil,$$

where $\lceil x \rceil$ denotes the ceiling function. The first few Bell numbers for $n = 1, 2, \ldots$ are

$$1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, \ldots$$

For each of the domains $DOM(v_i)$, the partitions π_i can be chosen independently. It follows that the search space for induced distinctions is equal to the combinations of all domain partitions:

Proposition 5 *Let QAP be a qualitative abstraction problem with finite domains* $DOM(\boldsymbol{v}) = DOM(v_1) \times \ldots \times DOM(v_n)$. *Then the size of the search space for induced distinctions* $\boldsymbol{\tau_{ind}}$ *is equal to*

$$\prod_{i=1}^{n} B_{SIZE(i)} \text{ where } SIZE(i) := |DOM(v_i)|.$$

Because Bell numbers grow over-exponentially with the number of elements in a set, it follows that the size of the search space for induced distinctions grows over-exponentially with the size of the domains of the variables.

5.5 Characterizing Induced Distinctions

We have seen that for a given qualitative abstraction problem, there always exists a domain abstraction that fulfills the definition of a distinguishing domain abstraction. But, is there always a unique maximal distinguishing abstraction, and, hence, a unique solution? Answers to this question are captured by the propositions in this section.

It turns out that the existence of a unique solution to a qualitative modeling problem $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ depends on properties of the set of external restrictions, minimality properties of R, and the relationship of observable and target distinctions.

5.5.1 The Scope of External Restrictions

First, we turn to the influence of the considered set of external restrictions. A qualitative abstraction problem is said to be *complete* if the set of possible external restrictions to be considered in definition 34 comprises all relations at the level of observable distinctions:

Definition 37 (Completeness of QAP) *Let $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ be a qualitative abstraction problem. QAP is complete, if each $R_{ext} \subseteq \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$ has to be considered as a possible external restriction.*

Note that QAP being complete does not imply that all variables in the model are observable. Instead, it states that each relation on the level of observable distinctions can occur as possible external restriction. The following proposition shows that completeness of the set of external restriction influences the uniqueness of a solution:

Proposition 6 (Non-unique Solution for Incomplete QAP) *For a qualitative abstraction problem that is not complete, the maximal distinguishing abstraction $\boldsymbol{\tau_{ind}}$ is in general not uniquely defined.*

We prove this proposition indirectly by giving an example of an incomplete qualitative abstraction problem that has more than one solution.

Example 9 (Non-unique Solution for Incomplete QAP) *Let $\boldsymbol{v} = (v_1, v_2, v_3)$. Let $DOM(v_i) = \{0, 1\}$ for $i = 1, 2, 3$. Let R express equality between v_1, v_2 and v_3, i.e.*

$$R(\boldsymbol{v}) = \{(0, 0, 0), (1, 1, 1)\}.$$

Let us assume that the non-trivial observable distinctions are

$$\pi_{obs,1} = \{\{0\}, \{1\}\}, \pi_{obs,2} = \{\{0\}, \{1\}\}$$

and that the only non-trivial target distinction is to determine whether v_3 is zero or not:

$$\pi_{targ,3} = \{\{0\}, \{1\}\}.$$

It follows that there are 16 possible external restrictions R_{ext}. For the subset of external restrictions

$$\{\{(\{0\},\{0\},\{0,1\})\},\{(\{1\},\{1\},\{0,1\})\},\{(\{0\},\{1\},\{0,1\}),(\{1\},\{0\},\{0,1\})\}\},$$

there are two induced distinctions, given by the partitions

$$\pi_{ind,1} = \{\{0\},\{1\}\},\ \pi_{ind,2} = \{\{0,1\}\},$$
$$\pi'_{ind,1} = \{\{0,1\}\},\ \pi'_{ind,2} = \{\{0\},\{1\}\}.$$

□

The two maximal distinguishing abstractions in example 9 reflect the fact that either v_1 or v_2 is not needed for determining the target distinction. The reason is that for the given set of external restrictions, a coarse distinction for one variable might be compensated by a fine distinction for another variable.

Consider the special case where observations are restricted to single tuples. This might occur, for instance, in the case where for each of the observable variables of a device, there is a fixed correspondence to measured sensor values. Example 10 shows that also in this case, there might be more than one induced distinction:

Example 10 (Non-unique Solution for Observations as Tuples) *Let the variables $\boldsymbol{v}$, $DOM(\boldsymbol{v})$, $R(\boldsymbol{v})$, $\boldsymbol{\pi_{obs}}$ and $\boldsymbol{\pi_{targ}}$ be given as in example 9. For the set of external restrictions that comprises all tuples from $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$*

$$R_{ext} \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$$

there are two maximal induced abstractions, given by the partitions

$$\pi_{ind,1} = \{\{0\},\{1\}\},\ \pi_{ind,2} = \{\{0,1\}\},$$
$$\pi'_{ind,1} = \{\{0,1\}\},\ \pi'_{ind,2} = \{\{0\},\{1\}\}.$$

□

Like in example 9, the two solutions reflect the fact that either v_1 or v_2 is not needed for determining the target distinction for variable v_3. In the following, we concentrate on qualitative abstraction problems that are complete.

5.5.2 The Scope of the Model Relation

Next, we investigate the influence of the model relation R, in particular, effects related to redundancy of domain values.

Definition 38 (Redundant Domain Value) *A domain value $val \in DOM(v_i)$ is called redundant if it is not part of any tuple of the model relation, i.e.*

$$val \notin \Pi_{v_i}(R).$$

The term *redundant* refers to the fact that such values can be removed from the corresponding domain without affecting the tuples of the model relation (see also [Tsa93]).

Definition 39 (Minimality of QAP) *A qualitative abstraction problem QAP is called minimal if none of the domains $DOM(v_i)$ contains redundant values.*

Minimality is a precondition for a unique solution to a qualitative abstraction problem. Otherwise, the maximal distinguishing abstraction is in general not uniquely defined:

Proposition 7 (Non-unique Solution for Non-Minimal QAP) *For a complete and non-minimal qualitative abstraction problem, the maximal distinguishing abstraction $\boldsymbol{\tau_{ind}}$ is in general not uniquely defined.*

The reason is that redundant parts of the domain of a variable can be allocated to different partition elements of an induced distinction without affecting the result of the abstraction. This is illustrated by the following example.

Example 11 (Non-unique Solution for Non-minimal QAP) *Let $\boldsymbol{v} = (v_1, v_2)$. Let $DOM(v_i) = \{0, 1, 2, 3\}$ for $i = 1, 2$. Let R be given by*

$$R(\boldsymbol{v}) = \{(0, 0), (0, 1), (1, 2), (1, 3)\}.$$

Variable v_1 has two redundant domain values. Assume that the non-trivial distinctions are given by the target distinction

$$\pi_{targ,1} = \{\{0, 1\}, \{2, 3\}\}, \pi_{targ,2} = \{\{0, 1\}, \{2, 3\}\}$$

and the observable distinction

$$\pi_{obs,1} = \{\{0, 2\}, \{1, 3\}\}.$$

There are two induced distinctions which differ with respect to the partition for variable v_1:

$$\pi_{ind,1} = \{\{0, 2\}, \{1, 3\}\}, \pi'_{ind,1} = \{\{0, 1\}, \{2, 3\}\}.$$

□

5.5.3 The Scope of Target and Observable Distinctions

Next, we investigate the influence of the observable distinctions and target distinctions. The relationship between target and observable partitions affects the granularity of the resulting abstraction. On the one hand, if the available observations are overly detailed for the given target distinctions, the model can be simplified because not all observations have to be distinguished from each other. If, on the other hand, there do not exist detailed enough observations to derive all

target distinctions, this can also give rise to abstractions. In this case, it would be unnecessary to reflect the target distinctions in the model that cannot be derived — in a sense "overriding" the inital desire to distinguish these solutions.

Hence, the situation where the target distinctions can indeed be distinguished based on the model and the external restrictions is of particular interest. The view we take here is that task-dependent abstraction is only adequate if the granularity level of observable distinctions is detailed enough such that the target distinctions actually come into effect. This case is captured by the following definition.

Definition 40 (Observability of QAP) *Let* $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ *be a qualitative abstraction problem. QAP is called observable, if*

$$\forall \boldsymbol{val} \in \boldsymbol{\tau_{targ}}(R(\boldsymbol{v})) : \exists R_{ext} \subseteq \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v})) \text{ s.th. } \boldsymbol{val} = \boldsymbol{\tau_{targ}}(R \bowtie R_{ext}).$$

A special case of observable QAP is the class of problems where $\boldsymbol{\tau_{obs}}$ is a refinement of $\boldsymbol{\tau_{targ}}$. This occurs e.g. for the pedal position sensor in examples 5 and 6, for which the target distinctions where given for a subset of the observable variables. It follows that for the pedal position sensor example, QAP is observable. Of course, we later on expect that it is the *program*, rather than the user, that tells us whether a specified QAP has this property. Observability is crucial for the uniqueness of solutions to a qualitative abstraction problem, as stated by the following proposition.

Proposition 8 (Non-unique Solution for Non-Observable QAP) *For a complete, minimal and non-observable qualitative abstraction problem, the maximal distinguishing abstraction* $\boldsymbol{\tau_{ind}}$ *is in general not uniquely defined.*

Example 12 (Non-unique Solution for Non-Observable QAP) *Let* $\boldsymbol{v} = (v_1, v_2)$. *Let* $DOM(v_i) = \{0, 1, 2\}$, $i = 1, 2$. *Let* R *be given by*

$$R(\boldsymbol{v}) = \{(0,0), (0,1), (0,2), (1,1), (1,2), (2,1), (2,2)\}.$$

Let us assume that the only non-trivial target distinction is to determine whether v_1 *is greater than one or not:*

$$\pi_{targ,1} = \{\{0,1\}, \{2\}\}.$$

For the observable distinctions

$$\pi_{obs,1} = \{\{0\}, \{1,2\}\}, \pi_{obs,2} = \{\{0,1,2\}\},$$

there are two induced distinctions, given by different partitions for v_1*:*

$$\pi_{ind,1} = \{\{0\}, \{1,2\}\}, \pi'_{ind,1} = \{\{0,1\}, \{2\}\}.$$

□

In example 12, the target partition for variable v_1 is not observable. Whenever the target partition element $P_{1,2} := \{2\}$ appears in a solution, the target partition element $P_{1,1} := \{0, 1\}$ also appears in this solution. Hence, for the induced distinctions, it is sufficient to distinguish the domain value 0 from the domain value 2 for variable v_1. The two maximal distinguishing abstractions $\boldsymbol{\pi_{ind}}$, $\boldsymbol{\pi'_{ind}}$ reflect the fact that this can be done in two ways.

5.5.4 The Scope of the Problem Solving Task

In this section, we capture the influence of the problem solving task on the induced distinctions. Chapter 3 introduced prediction and diagnosis as two basic model-based problem solving tasks.

For prediction, the considered external restrictions are assumed to be consistent with the model, and we are interested in deriving the resulting restrictions for other variables. In contrast, for the task of diagnosis, it is possible that the model is inconsistent with a given external restriction, such that it is necessary to perform a revision of the model.

Definition 41 (Consistency of QAP) *Let $QAP = (R, \tau_{obs}, \tau_{targ})$ be a qualitative abstraction problem. QAP is consistent, if all external restrictions are consistent with the model, i.e. $\tau_{obs}(R) = \tau_{obs}(DOM(\boldsymbol{v}))$.*

The condition that QAP is consistent is not a restriction, for a given behavior model can always be modified in such a way that all external restrictions are consistent with the model. The principle is to turn a diagnostic problem solving task into a prediction task by anticipating possible revisions in the model (see section 3.6.3), in the extreme case by adding unknown behavior modes that correspond to unrestricted behavior of components.

5.6 Induced Distinctions and Interchangeability

In this section, it is shown how the problem of finding interchangeable values in constraint satisfaction (section 4.6.1) is related to finding induced distinctions for a qualitative abstraction problem. The following proposition states that interchangeability can be reconstructed as a special case of a QAP.

Proposition 9 (Interchangeability as Qualitative Abstraction Problem) *Let $QAP = (R, \tau_{obs}, \tau_{targ})$ be a complete, minimal and observable qualitative abstraction problem where $\tau_{obs} = \tau_{id}$, $\tau_{targ} = \tau_{triv}$. Then the maximal distinguishing domain abstraction τ_{ind} for QAP is equal to $\tau_{FI,R}$.*

Proof. Because QAP is complete, the set of considered external restrictions contains in particular each tuple of $DOM(\boldsymbol{v})$ as possible R_{ext}. For each such R_{ext} that is inconsistent with R, $\tau_{ind}(R_{ext})$ must also remain inconsistent with $\tau_{ind}(R)$, and for each R_{ext} that is consistent with R, $\tau_{ind}(R_{ext})$ must also remain consistent with $\tau_{ind}(R)$. It follows that τ_{ind} must preserve exactly the tuples of R, and thus the distinctions in $\tau_{FI,R}$ are necessary for τ_{ind} to be distinguishing. On the other hand, if two domain values for a variable v_i are fully interchangeable, then there does not exist an external restriction that would yield a different result on the level of target distinctions if these domain values are not distinguished. Thus the distinctions in $\tau_{FI,R}$ are sufficient for τ_{ind} to be distinguishing. It follows that τ_{ind} is equal to $\tau_{FI,R}$. □

5.6.1 Complexity of Finding Induced Distinctions

Based on the results of the previous section, the following proposition states that finding induced distinctions for compositional models is an inherently hard problem.

Proposition 10 (Complexity of Finding Induced Distinctions) *Let $QAP = (R, \tau_{obs}, \tau_{targ})$ be a qualitative abstraction problem for a compositional behavior model R given by a system description. Then finding an induced distinction τ_{ind} for QAP is NP-hard.*

Proof. Consider again the case where $\tau_{obs} = \tau_{id}$, $\tau_{targ} = \tau_{triv}$, and QAP is complete, minimal and observable. From proposition 9, it follows that in this situation, determining the induced distinctions is equal to the problem of determining interchangeable values for the relation R defined by the constraint network of the system description. Determining interchangeable values for a constraint network is known to be a NP-hard problem ([WF97]). It follows that finding an induced distinction τ_{ind} for a qualitative abstraction problem is also NP-hard. □

5.7 Determining Induced Distinctions

In this section, we describe how solutions can be obtained for a qualitative abstraction problem. In the following, we assume that for a given qualitative abstraction problem $QAP = (R, \tau_{obs}, \tau_{targ})$, a solution τ_{ind} is given by the merge of an abstraction $\tau_{obs}{}'$ of τ_{obs} and an abstraction $\tau_{targ}{}'$ of τ_{targ}. It will be shown that in this case, and under the preconditions identified in the previous sections, a unique solution for QAP can be found.

First, in this situation, a maximal distinguishing domain abstraction contains at least the distinctions given by τ_{targ}:

Theorem 1 (Decomposition of Induced Distinctions) *Let QAP be a qualitative abstraction problem that is complete, minimal, and observable. If τ_{ind} is a maximal distinguishing domain abstraction for QAP, then τ_{ind} is a refinement of τ_{targ}, i.e. $\tau_{ind} = MERGE(\tau_{obs}{}', \tau_{targ})$.*

Proof. Assume the contrary, i.e. for a variable v_i, there is a partition element in $\tau_{ind,i}$ that contains values from more than one target partition element. Because QAP is observable, for each target partition element of v_i, there exists at least one external restriction R_{ext} for which the projection of $\tau_{targ}(R \bowtie R_{ext})$ on v_i is equal to the target partition element. For R_{ext}, the projection of $\tau_{targ}(\tau_{ind}(R) \bowtie \tau_{ind}(R_{ext}))$ on v_i contains more than one target partition element. It follows that $\tau_{ind}(R) \bowtie \tau_{ind}(R_{ext})$ yields a different solution at the level of target distinctions compared to $R \bowtie R_{ext}$. Hence, τ_{ind} cannot be a maximal distinguishing domain abstraction. □

5.7.1 Observation Partitions and Solution Partitions

Theorem 1 means that the target distinctions must be kept in the induced distinctions if QAP is observable. Hence, the source for abstraction of the induced distinctions can only be an abstraction of the observable distinctions.

The space for external restrictions is given by $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$. For each tuple $OBS_k \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$, define $R_{OBS,k}$ to be the join of the observation with the model relation:

$$R_{OBS,k} := R \bowtie OBS_k.$$

If QAP is complete, i.e. each $OBS_k \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$ is considered, the $R_{OBS,k}$ cover the model relation. Because the OBS_k are mutually disjoint, the $R_{OBS,k}$ are mutually disjoint. It follows that the $R_{OBS,k}$ form a partition of the model relation R:

Definition 42 (Observation Partition) *For a qualitative abstraction problem* $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$*, the set*

$$\Omega(R, \boldsymbol{\tau_{obs}}) := \bigcup_k \{R_{OBS,k}\}$$

is denoted observation partition.

Let further $R_{SOL,k}$ be the solution obtained for $R_{OBS,k}$ at the level of $\boldsymbol{\tau_{targ}}$:

$$R_{SOL,k} := \boldsymbol{\tau_{targ}}(R_{OBS,k}).$$

Then the $R_{OBS,k}$ that obtain the same solution, i.e. for which $R_{SOL,k}$ is equal, form a partition of the set $\Omega(R, \boldsymbol{\tau_{obs}})$. Let $R_{SOL,OBS,k}$ be defined by

$$R_{SOL,OBS,k} := \bigcup_{j:R_{SOL,j}=R_{SOL,k}} R_{OBS,j}.$$

Then the elements of the set of all such relations $R_{SOL,OBS,k}$ define a partition of the observation partition:

Definition 43 (Solution Partition) *For a qualitative abstraction problem* $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$*, the set*

$$\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}}) := \bigcup_k \{R_{SOL,OBS,k}\}$$

is denoted solution partition.

In other words, $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ defines a partition of the model relation that is an abstraction of the partition of the model relation defined by $\Omega(R, \boldsymbol{\tau_{obs}})$ (see figure 5.7). The following example illustrates the above definitions.

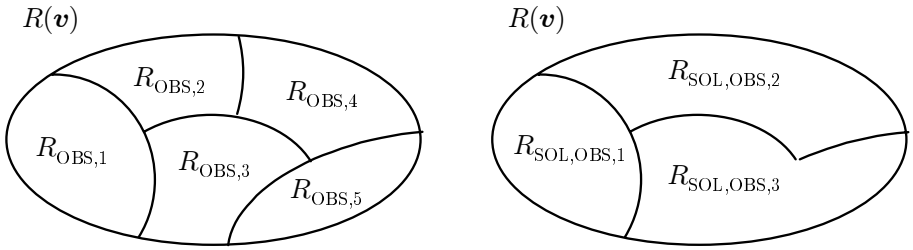

Figure 5.7: The elements of the observation partition (left) form a partition of the model relation, while the elements of the solution partition (right) form an abstraction of this partition

Example 13 (Equality) *Let* $\boldsymbol{v} = (v_1, v_2)$. *Let* $DOM(v_i) = \{0, 1, 2\}$ *for* $i = 1, 2$. *Let* R *express the behavior* $v_1 = v_2$, *i.e.*

$$R(\boldsymbol{v}) = \{(0,0), (1,1), (2,2)\}.$$

Assume that the only non-trival observable partition is a partition for v_1

$$\pi_{obs,1} = \{\{0\}, \{1\}, \{2\}\}, \pi_{obs,2} = \{\{0, 1, 2\}\}$$

and that the only non-trival target partition is given by a partition for v_2*:*

$$\pi_{targ,1} = \{\{0, 1, 2\}\}, \pi_{targ,2} = \{\{0\}, \{1, 2\}\}.$$

Then the set $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$ *contains the elements*

$$\begin{aligned} OBS_1 &= (\{0\}, \{0, 1, 2\}), \\ OBS_2 &= (\{1\}, \{0, 1, 2\}), \\ OBS_3 &= (\{2\}, \{0, 1, 2\}) \end{aligned}$$

and the set $\Omega(R, \boldsymbol{\tau_{obs}})$ *contains the three elements*

$$\begin{aligned} R_{OBS,1} &= \{(0,0)\}, \\ R_{OBS,2} &= \{(1,1)\}, \\ R_{OBS,3} &= \{(2,2)\}. \end{aligned}$$

Further,

$$\begin{aligned} R_{SOL,1} = \{(\{0,1,2\}, \{0\})\}, R_{SOL,OBS,1} = \{(0,0)\}, \\ R_{SOL,2} = \{(\{0,1,2\}, \{1,2\})\}, R_{SOL,OBS,2} = \{(1,1), (2,2)\}, \\ R_{SOL,3} = \{(\{0,1,2\}, \{1,2\})\}, R_{SOL,OBS,3} = \{(1,1), (2,2)\} \end{aligned}$$

and the set $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ *contains the two elements (see also figure 5.8)*

$$R_{SOL,OBS,1} = \{(0,0)\}, R_{SOL,OBS,2} = \{(1,1), (2,2)\}.$$

□

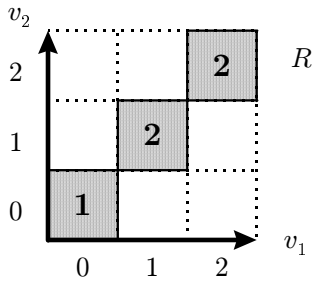

Figure 5.8: The two elements $R_{SOL,OBS,1}$ and $R_{SOL,OBS,2}$ of the partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ for example 13 (numbers denote the indices of the partition elements)

5.7.2 Approximations to Induced Distinctions

The following theorem provides a sufficient, but not necessary condition for induced distinctions.

Theorem 2 (Sufficient Condition) *Let QAP be a qualitative abstraction problem that is complete and minimal. If* $\boldsymbol{\tau_{ind}}$ *is a maximal distinguishing domain abstraction for QAP, then* $\tau_{ind,i}$ *is a refinement of any domain partition* $\pi_i = \{P_{i,1}, P_{i,2}\}$ *given by*

$$P_{i,1} = \Pi_{v_i}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS})), P_{i,2} = DOM(v_i) \backslash P_{i,1}, R_{SOL,OBS} \in \Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}}).$$

Proof. The case where either $P_{i,1} = \emptyset$ or $P_{i,2} = \emptyset$ is trivial. Now assume the case where the partition elements $P_{i,1}$ and $P_{i,2}$ are both non-empty, and let $\boldsymbol{\tau_{obs}}(R_{SOL,OBS})$ be the element of $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ used to obtain π_i. Consider the two relations

$$R_{ext} := P_{i,1}, R'_{ext} := P_{i,2}.$$

R_{ext} and R'_{ext} are at the level of granularity of the observable distinctions, and because QAP is complete, R_{ext} and R'_{ext} can occur as possible external restrictions. Then $\boldsymbol{\tau_{targ}}(R \bowtie R_{ext}) =: SOL$, i.e. the external restriction R_{ext} yields the solution SOL at the level of the target distinctions. Likewise, $\boldsymbol{\tau_{targ}}(R \bowtie R'_{ext}) =: SOL'$, i.e. the external restriction R'_{ext} yields a solution SOL'. Because QAP is minimal, both $SOL \neq \emptyset$ and $SOL' \neq \emptyset$. Observe further that since $P_{i,2}$ is inconsistent with $\boldsymbol{\tau_{obs}}(R_{SOL,OBS})$, the solution obtained for R_{ext} is different from the solution obtained for R'_{ext}, i.e. $SOL \neq SOL'$. Assume that the domain partition corresponding to $\boldsymbol{\tau_{ind}}$ combines $P_{i,1}$, $P_{i,2}$ in one partition element. Then

$$\boldsymbol{\tau_{targ}}(\boldsymbol{\tau_{ind}}(R) \bowtie \boldsymbol{\tau_{ind}}(R_{ext}) \supseteq SOL \cup SOL',$$
$$\boldsymbol{\tau_{targ}}(\boldsymbol{\tau_{ind}}(R) \bowtie \boldsymbol{\tau_{ind}}(R'_{ext}) \supseteq SOL' \cup SOL.$$

Because $SOL \neq SOL'$ and $SOL, SOL' \neq \emptyset$, it holds that $SOL \cup SOL' \neq SOL$ or $SOL \cup SOL' \neq SOL'$. Hence, for at least one of the external restrictions R_{ext} or R'_{ext}, $\boldsymbol{\tau_{ind}}$ derives a different result on the level of target distinctions compared to the base model relation. Therefore, $\boldsymbol{\tau_{ind}}$ cannot be a distinguishing domain abstraction, and it follows that any $\boldsymbol{\tau_{ind}}$ must spearate $P_{i,1}$ from $P_{i,2}$, which means that it is a refinement of the domain partition π_i. □

Example 14 (Equality) *Consider again example 13. By applying theorem 2, the following partition elements are obtained for variable* v_1

$$P_{1,1,1} := \Pi_{v_1}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS,1})) = \{0\}, P_{1,2,1} := DOM(v_1) \setminus P_{1,1,1} = \{1,2\},$$
$$P_{1,1,2} := \Pi_{v_1}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS,2})) = \{1,2\}, P_{1,2,2} := DOM(v_1) \setminus P_{1,1,2} = \{0\},$$

and the following partition elements are obtained for variable v_2 :

$$P_{2,1,1} := \Pi_{v_1}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS,1})) = \{0,1,2\}, P_{1,2,1} := DOM(v_1) \setminus P_{2,1,1} = \emptyset,$$
$$P_{2,1,2} := \Pi_{v_1}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS,2})) = \{0,1,2\}, P_{2,2,2} := DOM(v_1) \setminus P_{2,1,2} = \emptyset.$$

By applying also theorem 1 (see also figure 5.9), it follows that a maximal distinguishing domain abstraction $\boldsymbol{\tau_{ind}}$ *must be a refinement of*

$$\pi_1 = \{\{0\},\{1,2\}\}, \pi_2 = \{\{0\},\{1,2\}\}.$$

□

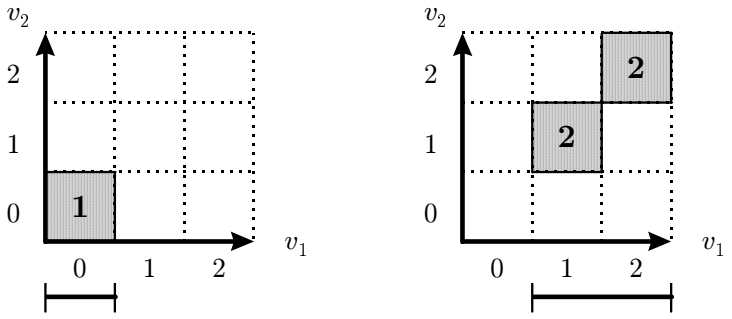

Figure 5.9: Projection of $R_{SOL,OBS,1}$ (left) and $R_{SOL,OBS,2}$ (right) on variable v_1 yields a distinction between the domain values for example 14

For the example above, the distinctions derived by theorems 1 and 2 happened to be sufficient and necessary in order to obtain a maximal distinguishing domain abstraction. In contrast, the following example presents a qualitative abstraction problem for which the approximation derives only necessary, but not sufficient distinctions.

Example 15 (Xor-Gate) *Let* $\boldsymbol{v} = (v_1, v_2, v_3)$. *Let* $DOM(v_i) = \{0, 1\}$ *for* $i = 1, 2, 3$. *Let* R *express the behavior* $v_3 = XOR(v_1, v_2)$, *i.e.*

$$R(\boldsymbol{v}) = \{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}.$$

Let us assume that a non-trivial observable distinction is given only for variables v_1 *and* v_2 *whose granularity is equal to their base domain, and that the only non-trivial target distinction is to determine whether* v_3 *is zero or not:*

$$\pi_{obs,1} = \{\{0\}, \{1\}\}, \pi_{obs,2} = \{\{0\}, \{1\}\}, \pi_{targ,3} = \{\{0\}, \{1\}\}.$$

Then $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$ *has four elements, and the set* $\Omega(R, \boldsymbol{\tau_{obs}})$ *contains the four elements*

$$\begin{aligned} R_{OBS,1} &= \{(0,0,0)\}, \\ R_{OBS,2} &= \{(0,1,1)\}, \\ R_{OBS,3} &= \{(1,0,1)\}, \\ R_{OBS,4} &= \{(1,1,0)\}. \end{aligned}$$

Further,

$$\begin{aligned} R_{SOL,1} &= \{(\{0,1\}, \{0,1\}, 0)\}, R_{SOL,OBS,1} = \{(0,0,0), (1,1,0)\}, \\ R_{SOL,2} &= \{(\{0,1\}, \{0,1\}, 1)\}, R_{SOL,OBS,2} = \{(0,1,1), (1,0,1)\}, \\ R_{SOL,3} &= \{(\{0,1\}, \{0,1\}, 1)\}, R_{SOL,OBS,2} = \{(0,1,1), (1,0,1)\}, \\ R_{SOL,4} &= \{(\{0,1\}, \{0,1\}, 0)\}, R_{SOL,OBS,1} = \{(0,0,0), (1,1,0)\}, \end{aligned}$$

and, hence, the set $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ *contains the two elements*

$$R_{SOL,OBS,1} = \{(0,0,0), (1,1,0)\}, R_{SOL,OBS,2} = \{(0,1,1), (1,0,1)\}.$$

The application of theorem 2 yields only the trivial partition as a lower bound for the granularity of v_1 *and* v_2 *(see also figure 5.10). However, the maximal distinguishing domain abstraction for the example is given by the granularity of the base domain:*

$$\pi_{ind,1} = \{\{0\}, \{1\}\}, \pi_{ind,2} = \{\{0\}, \{1\}\}.$$

□

5.7.3 Complete Solution to Induced Distinctions

The following theorem provides the complete solution for the problem of deriving induced distinctions.

Theorem 3 (Sufficient and Necessary Condition) *Let QAP be a qualitative abstraction problem that is complete, minimal, observable and consistent. Then* $\boldsymbol{\tau_{ind}}$ *is equal to the merge of the target distinctions with any domain partition* π_i *given by*

$$\tau_{FI,\Lambda} \text{ where } \Lambda := \boldsymbol{\tau_{obs}}(R_{SOL,OBS}),\ R_{SOL,OBS} \in \Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}}).$$

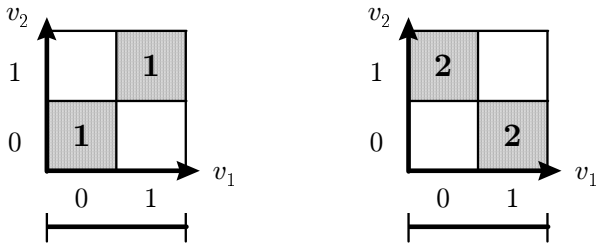

Figure 5.10: Projection of $R_{SOL,OBS,1}$ (left) and $R_{SOL,OBS,2}$ (right) on variable v_1 yields no distinction for example 15

Proof. The construction is similar to the proof of theorem 2. Let $R_{SOL,OBS}$ be an element of $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ used to obtain π_i. Let $P_{i,1}, P_{i,2}$ be two partition elements of π_i. Consider the two relations

$$R_{ext} := \sigma_{v_i \in P_{i,1}}(\boldsymbol{\tau_{obs}}(R_{SOL,OBS})),$$
$$R'_{ext} := \{val\} \times \Pi_{\boldsymbol{v} \setminus v_i}(R_{ext}) \text{ where } val \in P_{i,2}.$$

R_{ext} and R'_{ext} are at the level of granularity of the observable distinctions. Because QAP is complete, R_{ext} and R'_{ext} can occur as possible external restrictions. Then $\boldsymbol{\tau_{targ}}(R \bowtie R_{ext}) =: SOL$, i.e. the external restriction R_{ext} yields the solution SOL at the level of the target distinctions. Likewise, $\boldsymbol{\tau_{targ}}(R \bowtie R'_{ext}) =: SOL'$, i.e. the external restriction R'_{ext} yields a solution SOL'. Because QAP is consistent, both $SOL \neq \emptyset$ and $SOL' \neq \emptyset$. Now, because $P_{i,1}$ is not fully interchangeable for $P_{i,2}$ with respect to the relation $\boldsymbol{\tau_{obs}}(R_{SOL,OBS})$, $R_{ext} \cap R'_{ext} = \emptyset$, and thus R_{ext}, R'_{ext} yield different solutions, i.e. $SOL \neq SOL'$. Then similar to the proof of theorem 2, a domain abstraction $\boldsymbol{\tau_{ind}}$ that combines $P_{i,1}$, $P_{i,2}$ in one partition element yields

$$\boldsymbol{\tau_{targ}}(\boldsymbol{\tau_{ind}}(R) \bowtie \boldsymbol{\tau_{ind}}(R_{ext}) \supseteq SOL \cup SOL',$$
$$\boldsymbol{\tau_{targ}}(\boldsymbol{\tau_{ind}}(R) \bowtie \boldsymbol{\tau_{ind}}(R'_{ext}) \supseteq SOL' \cup SOL.$$

It holds that $SOL \cup SOL' \neq SOL$ or $SOL \cup SOL' \neq SOL'$, which means that for at least one of the external restrictions R_{ext} or R'_{ext}, $\boldsymbol{\tau_{ind}}$ derives a different result on the level of target distinctions compared to the base model relation. Hence, the distinctions in π_i are necessary for $\boldsymbol{\tau_{ind}}$ to be a distinguishing domain abstraction. Conversely, for values that are fully interchangeable for all $\boldsymbol{\tau_{obs}}(R_{SOL,OBS})$ with $R_{SOL,OBS} \in \Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$, then because $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ comprises all possible external restrictions from $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$, it follows that no external restriction exists that would yield a different solution if these values are abstracted. Therefore, the distinctions given by the domain partitions π_i and the target distinctions are sufficient for $\boldsymbol{\tau_{ind}}$ to be a distinguishing domain abstraction. □

Example 16 (Xor-Gate) *Consider again example 15. Then for variable v_1,*

$$\Pi_{v_2,v_3}(\sigma_{v_1=0}(R_{SOL,OBS,1})) = \{(0,0)\},$$
$$\Pi_{v_2,v_3}(\sigma_{v_1=1}(R_{SOL,OBS,1})) = \{(1,0)\},$$
$$\Pi_{v_2,v_3}(\sigma_{v_1=0}(R_{SOL,OBS,2})) = \{(1,1)\},$$
$$\Pi_{v_2,v_3}(\sigma_{v_1=1}(R_{SOL,OBS,1})) = \{(0,1)\}.$$

By applying theorem 3, the following partition elements are obtained for variable v_1:

$$\pi_1 = \{\{0\},\{1\}\}.$$

Conversely, for variable v_2,

$$\Pi_{v_1,v_3}(\sigma_{v_2=0}(R_{SOL,OBS,1})) = \{(0,0)\},$$
$$\Pi_{v_1,v_3}(\sigma_{v_2=1}(R_{SOL,OBS,1})) = \{(1,0)\},$$
$$\Pi_{v_1,v_3}(\sigma_{v_2=0}(R_{SOL,OBS,2})) = \{(1,1)\},$$
$$\Pi_{v_1,v_3}(\sigma_{v_2=1}(R_{SOL,OBS,2})) = \{(0,1)\},$$

and the following partition elements are obtained for variable v_2:

$$\pi_2 = \{\{0\},\{1\}\}.$$

It follows that the maximal distinguishing domain abstraction for this qualitative abstraction problem is

$$\pi_{ind,1} = \{\{0\},\{1\}\}, \pi_{ind,2} = \{\{0\},\{1\}\}, \pi_{ind,3} = \{\{0\},\{1\}\}.$$

□

The previous sections have identified different properties whose absence can lead to non-unique solutions of a qualitative abstraction problem. As a consequence of theorem 3, it follows that these properties are complete in the sense that if each of them holds, there is a unique solution to a qualitative abstraction problem.

The approximation and the complete condition can be considered as extreme positions in a spectrum of definitions that vary in the granularity the elements of the solution partition are distinguished from each other. While theorem 2 considers only differences taking into account external restrictions that restrict a single variable, theorem 3 considers differences taking into account external restrictions that possibly restrict all of the variables. This coincides with the intuitive idea that domain values have to be distinguished if either the domain values themselves already lead to different solutions (corresponding to theorem 2), or the domain values lead to different solutions if combined with additional restrictions for other variables (corresponding to theorem 3). The latter case is more general, i.e. theorem 3 obtains at least the distinctions obtained by theorem 2. However, for a given QAP, theorem 2 does not require all external restrictions to be consistent. Hence, it can be applied to more cases than theorem 3.

5.8 Discussion

Theorem 2 and theorem 3 capture the solution to our problem to derive task-dependent qualitative values. If we accept the view that induced distinctions capture much of the intuitive goal of deriving qualitative values, the results in this section can be seen as principled solutions to the problem of qualitative domain abstraction.

Chapter 4 has shown that in contrast, most approaches to qualitative reasoning assume a given level of abstraction, e.g. the sign domain. Sometimes, a specific level of abstraction is even hard-wired into the reasoning process itself. However, the results of this chapter illustrate that the characteristics of a problem solving task, in terms of what needs to be distinguished and what can be distinguished, have a crucial influence on the required model granularity. Unless such task-dependent requirements are made explicit, one can only resort to a model granularity that must fit all purposes and thus tends to be of limited use for a specific task.

There exist approaches in model-based reasoning, in particular in the field of model-based diagnosis, that include a limited notion of observable distinctions by assuming a separation of the system variables v_i into variables that can be observed — termed *observables* — and variables that cannot be observed — termed *non-observables* (see e.g. [CDD$^+$00]). The observable partitions introduced in this chapter are a more general means to express such kind of knowledge about the task: the above separation corresponds to a special case of an observable partition $\tau_{obs,i}$ that is equal to $\tau_{id,i}$ for the observables and equal to $\tau_{triv,i}$ for non-observables.

However, it should be pointed out that our notion of task requirements is still limited in the sense that it does not cover all intuitive notions of observable and target distinctions. In our framework, all distinctions have to be expressed in terms of domain abstractions.

One could argue, in particular, whether this is adequate for the specification of observable distinctions. For instance, for infinite domains, the observability of a variable might rather be given as a certain environment around a measured value, reflecting accuracy of the measurement. This does not correspond to a partition of the domain values of the variable. However, at least in some situations, this case could be handled by re-formulating the behavior model in terms of additional variables that express the deviation of a variable from its absolute value (see section 4.3.2) and stating observable distinctions for these deviations. But in general, more work would be necessary to extend the notion of task requirements to such cases.

Struss ([Str90]) presents a framework for analyzing properties of given qualitative reasoning methods, such as soundness and completeness (see chapter 4). Compared to this work, task-dependent abstraction is an "inverse" approach that aims at deriving abstractions that feature certain properties, such as being distinguishing and maximal. This constructive view allows us to analyze fundamental properties of the problem of deriving qualitative distinctions, e.g. in terms of

the existence of solutions. Whereas [Str90] focusses on interval-based representations, we are instead concerned with arbitrary domain abstractions. Another difference is that [Str90] does not incorporate a notion corresponding to target or observable distinctions and that it is considered only with the problem-solving task of prediction.

An earlier approach to deriving induced distinctions, described in [SS99], did not distinguish between the two different problem solving tasks of diagnosis vs. behavior prediction, and the two different types of distinctions, target distinctions vs. observable distinctions. As a consequence, it suffered from the problem that the granularity of the derived distinctions was dominated by the diagnosis task, for which it has been shown in section 5.6 that the resulting level of granularity corresponds to interchangeability of domain values. This illustrates that the separation of different problem solving tasks is essential, and that the separation of observable and target distinctions is essential.

Proposition 9 and theorem 3 reveal fundamental relationships between qualitative reasoning and interchangeability as a technique originating in constraint satisfaction to structure problems and compactly describe their solutions. However, note that interchageability is concerned only with possibilities for abstraction within a single CSP. In contrast, the motivation for task-dependent model abstractions is different: we are concerned with possibilites for abstraction given a *set* of CSPs that is implicitly defined by a model relation and set of external restrictions, under a given observable and target granularity. As shown in section 5.6, interchangeability is subsumed as a special case of task-dependent qualitative abstraction for which the observable distinction is equal to the base granularity, and the target distinction is to determine only whether the result is consistent or not.

Freuder and Choueiry ([Fre91], [FS95], and [CN98]) already observe that interchangeability is related to abstraction and the formation of "semantic groupings" within the domains of variables. Freuder ([Fre91]) notes that interchangeability can be viewed as a "concept formation process" that creates equivalence classes of values. The results of this chapter can be perceived as generalizing these ideas and putting them onto a firmer ground.

The results of this chapter provide the starting point for computing task-dependent model abstractions for a compositional behavior model. As has been shown, the solution partition, corresponding to a 2–level partition of the model relation, contains all the relevant information for deriving induced distinctions for a qualitative abstraction problem. In the next chapter, we will develop an algorithm that is able to exploit the ontological level of the behavior model in order to determine the solution partition. This means that the complexity of deriving a solution will be bound to structural features of the behavior model. This will give an answer to the last question raised at the end of section 5.3.

5.9 Summary

This chapter complemented the model-based reasoning framework that has been developed in chapter 3 by incorporating two task-dependent sources of qualitative abstraction: observable distinctions and target distinctions. The former reflects distinctions that cannot be made due to the granularity of the given observations, whereas the latter corresponds to distinctions that need not to be made due to the granularity of the required solutions. This allowed us to formally characterize our initial goal of deriving a qualitative model for a particular task as the problem of finding an abstraction level of a behavior model that incorporates only the necessary and sufficient distinctions in the domains of variables. The solution to this qualitative abstraction problem is captured by the notion of induced distinctions, which correspond to maximal distinguishing domain abstractions.

The second part of this chapter investigated fundamental properties of induced distinctions, such as uniqueness, and the theoretical (worst-case) complexity of deriving induced distinctions for a compositional behavior model. This analysis revealed principled features of qualitative domain abstraction. It has also been shown how the problem of determining interchangeability of values in constraint satisfaction can be viewed as a special case of a qualitative abstraction problem.

Chapter 6

Computation of Qualitative Model Abstractions

In this chapter, we describe how a solution to a qualitative abstraction problem QAP can be computed, given a system description defining a compositional behavior model, and a description of the task in terms of observable and target distinctions.

Chapter 5 has shown that under a number of preconditions for a QAP, induced distinctions can be derived based on the solution partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$. This chapter deals with the computation of the solution partition for the case of finite domains, and checking (or establishing) the necessary preconditions in order to apply theorems 1, 2 and 3.

An important part of the method is the representation of the model relation R. This is accomplished by an acyclic, hierarchical data structure called SD Tree. The SD Tree avoids representing the model relation for a system description explicitly, while guaranteeing that tuples of the model relation can be found in polynomial time in the size of the structure.

The computation of the solution partition is based on a propagation algorithm that operates on the SD Tree. This is the basis for efficient computation of induced distinctions for a compositional behavior model defined by a system description. The worst-case complexity of computing the solution depends on structural features of the model. More precisely, it allows to exploit the fact that for engineered devices, it is typically the case that not every variable directly constrains every other variable. This is important in order to scale the solution up to real-world problems like the example that will be presented in chapter 8.

6.1 Building Blocks for the Computation of Induced Distinctions

The sections of this chapter deal with the problem of how to compute induced distinctions $\boldsymbol{\tau_{ind}}$ for a composed system description based on the theorems in chapter 5. Considerable attention has to be paid to efficiency of solutions, as the

base model defined in the system description can be very large in terms of the number of variables and sizes of the domains. In particular, large domain sizes of the base model can occur if the base model is derived from a real-valued behavior model (see the examples in chapters 7 and 8). The computation of $\boldsymbol{\tau_{ind}}$ for a qualitative abstraction problem involves the subproblems of

(1) constructing the model relation R,

(2) checking (or establishing) the preconditions for applying the theorems, i.e. whether QAP is minimal, observable, and consistent in the case where a complete solution is required,

(3) computing the solution partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$.

We show how each step can be carried out based on efficient propagation algorithms. Finally, we deal with the problem of applying the derived abstraction $\boldsymbol{\tau_{ind}}$ to the original model of the system in order to get a transformed system description.

6.2 Computation of Model Relations

Determining the model relation can in principle be done by computing the join of the relations RC_i for the individual components C_i in the system description, i.e.

$$R(\boldsymbol{v}) = RC_1 \bowtie RC_2 \bowtie \ldots \bowtie RC_n.$$

However, $\boldsymbol{v}$ can contain hundreds of variables (see e.g. the example in chapter 8), and thus the model relation $R(\boldsymbol{v})$ may be very large. The number of tuples in $R(\boldsymbol{v})$ grows, in worst case, exponentially with the number of the variables and the size of the domains.

A basic representation for finite relations are ordered binary decision diagrams (OBDDs, see [Bry92]). The principle is to encode and manipulate relations using a symbolic representation of their characteristic function. Using OBDDs, it is often possible to obtain compact representations of constraint types, in particular when the degree of "constrainedness" is low or high. However, as [Bry92] points out, there are also examples of constraint types for which no compact representation as an ordered binary decision diagram can be found. It is an open research problem how feasible types of constraints can be characterized.

E.g. for the example presented in chapter 8, we found in our experiments on a PC with AMD Athlon with 700 MHz, 128 MB Ram, and Windows NT 4.0 that building an OBDD representation of R lead to a memory overflow after the first 20–25 variables, which corresponds to a relation in the order of roughly 10 billion tuples. While this number seems quite large, it still insufficient to represent the base model in this example.

Thus, a basic requirement is to avoid having to represent the relation R explicitly. Rather, we need a representation of R that is *implicit*, but still allows

to efficiently determine the elements of the solution partition and to check the preconditions for applying the theorems of chapter 5, e.g. whether redundant domain values occur for R. This has to be based on a systematic exploitation of model-specific features, rather than a method that may "sometimes" lead to a feasible representation.

6.2.1 Computation of Model Relations using Solution Synthesis Methods

In this section, we describe an efficient method to implicitly represent a relation R that is given by a system description. The representation is based on techniques known as decomposition and solution synthesis in constraint satisfaction ([DP88, Tsa93, Fre94, WF99]). The idea is to start with the space of all solutions, and to prune it incrementally by considering successively larger subproblems of the CSP. A subproblem of a CSP induced by a set of variables ([Fre91]) is the part of the CSP that directly constrains the set of variables:

Definition 44 (Induced Subproblem of a CSP) *Let S be a subset of the variables $(v_1, v_2 \ldots, v_n)$ of a CSP. The subproblem of the CSP induced by S consists of all the constraints whose schemes intersect with S.*

If all constraints have eventually been considered, the remaining solutions are the solutions of the original CSP. The advantage of this method is that early pruning of the search space — and therefore improvements in space and time — can be achieved if suitable subproblems can be identified that rule out significant portions of the solution space.

Decomposition of a CSP into subproblems is straightforward if its structure corresponds to a tree. However, in the case of a CSP where each subproblem is equal to the complete CSP, i.e. if each variable directly constrains each of the other variables in the system, solution synthesis becomes equal again to explicit determination of the solutions. Hence, a precondition for applying this method is that the original problem can be decomposed into a set of subproblems relatively easily.

For CSPs corresponding to system descriptions, subproblems can be interpreted as parts of the behavior model. For engineered devices, it is typical that interactions will be mediated through several components, such as defined interfaces, buses, or supplies. It is typically not the case that every variable directly affects each of the other variables. Hence, in a CSP corresponding to the behavior model e.g. of an automotive system, subproblems that are significantly smaller than the complete CSP can be expected to occur. Solution synthesis is especially suited for compositional problem descriptions, as it allows to exploit their specific structure in terms of subproblems and their interactions.

Exploitation of problem-specific features receives growing interest in the area of constraint satisfaction ([Tsa93],[Fre94]). The structural restriction of CSPs that each variable is constrained by a limited set of other variables only can be

captured by the concept of *nearly-acyclic* CSPs. The basic idea is that nearly-acyclic CSPs can be decomposed into acyclic (i.e. tree-structured) CSPs with limited effort, making them efficiently solvable. This technique originates from database theory, based on the fact that queries for relational databases can be interpreted as constraint satisfaction problems ([GLS00]). Structural decomposition has been studied mostly independent of the origin of the resulting CSP, e.g. for randomly generated CSPs ([WF99]). Here, it is directly motivated by the application domain of engineered devices.

6.2.2 Representing System Descriptions as Constraint Graphs

Applying decomposition and solution synthesis techniques to system descriptions requires some notational prerequisites. They operate on a representation of a CSP as a graph:

Definition 45 (Graph) *A graph is a tuple $G = (V, E)$ where V is a set of nodes and E is a set of arcs that connect nodes. G is called a hypergraph if there exist arcs in E that connect more than two nodes.*

Graph representations for CSPs can be constructed in two ways, either as a primal constraint graph or as a dual constraint graph. In a primal constraint graph, nodes represent variables and arcs are associated with the sets of nodes residing in the same constraint. A binary CSPs can be associated with a primal constraint graph where arcs connect pairs of constrained variables. For a non-binary CSP, the primal constraint graph corresponds to a hypergraph.

Conversely, a dual constraint graph represents each constraint by a node (often termed *meta-variable* in this context) and associates a labeled arc with any two nodes that share variables. The arcs are labeled by the shared variables. Thus, a dual constraint graph representation transforms any CSP to a special type of binary CSP, where the domain of the meta-variables ranges over all value combinations permitted by the corresponding constraint, and adjacent meta-variables are restricted by equality constraints stating that their shared variables must have the same values.

Definition 46 (Meta-Variable) *A meta-variable corresponds to a subset $S = (v_{i_1}, v_{i_2} \ldots, v_{i_k})$ of variables of a CSP. The domain values of the meta-variable are tuples of the relation $DOM(v_{i_1}) \times DOM(v_{i_2}) \times \ldots \times DOM(v_{i_k})$ that are solutions of the subproblem induced by S.*

A meta-variable can be viewed as an extension of the concept of a variable. While the domain of a variable is given by a set of values, the domain of a meta-variable is given by a relation, and while domain values of a variable are single values, domain values of a meta-variable are tuples (also called *compound labels* in [Tsa93]). Based on this analogy of meta-variables and relations, we will use the terminology devised for relations also for meta-variables. In particular, the scheme of a meta-variable denotes the set S of its variables.

A special type of dual constraint graph is the dual constraint graph that represents a system description for a behavior model:

Definition 47 (SD Graph) *An SD Graph is a dual constraint graph representation for a system description SD, where the constraints are given by the component behavior descriptions RC_i defined in SD, and the variables are given by the variables defined in SD.*

The advantages using dual constraint graphs to represent the CSPs corresponding to system descriptions are two-fold. First, we can retain the problem-specific topology of the system description in terms of components and their connections. Second, we can employ methods developed for binary CSPs in order to solve a non-binary CSP. In particular, if the constraint graph of a CSP is equal to a tree, there exist efficient algorithms in order to obtain its solutions. The first step in order to obtain the model relation R is therefore to build a dual constraint graph representation for the system description. This is accomplished by the following algorithm:

Algorithm (Transformation of System Description to SD Graph)

Step 1 For each component in the system description, build a meta-variable involving all variables for this component (including mode variables, if necessary).

Step 2 For each pair of variables in the system description that are identified with each other, establish an arc between the meta-variables that contain the two variables. The arc represents the binary constraint that the pair of variables must be equal.

The result is a dual constraint graph representation for the system description. Figure 6.1 shows an example of an SD Graph for the pedal position sensor model presented in chapter 3.

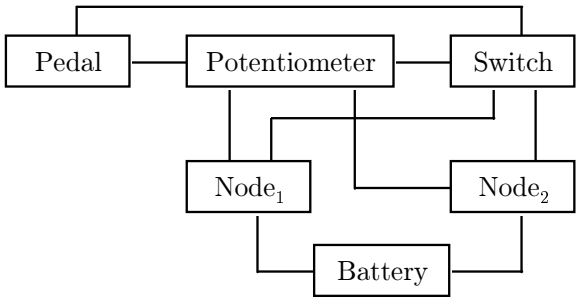

Figure 6.1: SD Graph for Model of the Pedal Position Sensor

6.2.3 Hierarchical Clustering of System Descriptions

The basic idea to derive solutions for a CSP representing a system description is that for a tree-structured constraint graph, the set of solutions can be derived in polynomial time using local constraint filtering techniques ([DP88]). Hence, the aim is to transform the constraint graph into a tree representation, even if the original constraint graph representation of the problem does not correspond to a tree. For a dual constraint graph, this can be achieved by systematically forming larger clusters (i.e. new meta-variables) from the original CSP and arranging them hierarchically in the form of a tree. The obtained meta-variables can be organized into a hierarchical structure that is called a *synthesis tree* (see [Fre78], [Tsa93]):

Definition 48 (Synthesis Tree) *A synthesis tree of a CSP represented by a constraint graph is a tree whose leaf nodes are the nodes of the constraint graph, and the intermediate nodes are meta-variables corresponding to the combination of their children. A synthesis tree is called minimal, iff each meta-variable represents only value combinations that are consistent with all constraints of the CSP.*

A special type of synthesis tree is the synthesis tree that is obtained for a CSP derived from a system description. This synthesis tree will be called *SD Tree*:

Definition 49 (SD Tree) *A SD Tree is the synthesis tree obtained from a SD Graph.*

In a SD Tree, the meta-variables can be interpreted as clusters (i.e. super-components) formed from the ontological elements (i.e. components) of the original system description. A minimal SD Tree implicitly represents the model relation R for the corresponding system description.

Meta-variables in a synthesis tree have domains that are based on the domains of the variables that constitute them. The domains of meta-variables thus potentially grow in a combinatorial way with the number of ground variables in the tree below them. Hence, the amount of information to be stored in order to represent the meta-variables varies with the topology of the tree. For a given system description, there may be many different possible SD Trees, corresponding to different ways of forming super-components from the ontological elements. Strategies for finding good synthesis tree topologies will be presented in section 6.2.5. Automatically building a SD Tree from a SD Graph involves two subproblems: computation of the meta-variables in order to form the nodes of the SD Tree, and constraint satisfaction based on the SD Tree in order to minimize it. This will be described in the following two sections.

6.2.4 Building SD Trees

Clustering, i.e. the generation of a synthesis tree from a constraint graph, involves to recursively determine subproblems in the constraint graph. This is accomplished by repeatedly identifying cliques in the constraint graph, and building

a new meta-variable for a clique by eliminating the arcs between the nodes in the clique. The nodes of the clique then become the children of the new meta-variable in the tree. This process is repeated until there are no more arcs left. For the special case where the constraint graph is a SD Graph, the following algorithm accomplishes the generation of the SD Tree:

Algorithm (Generation of SD Tree from SD Graph)

Step 1 Identify cliques in the SD Graph.

Step 2 Build a new meta-variable y for a selected clique $x_1, x_2, \ldots, x_k$ connected by arcs labeled with the set of variables S. Specify the scheme of y as the union of the schemes of the meta-variables in the clique without the variables in S:

$$scheme(y) = \bigcup_{i=1,\ldots,k} scheme(x_i) \backslash S.$$

Specify the domain values of y as the tuples in the join of the domains of the meta-variables in the clique, projected on the variables in $scheme(y)$:

$$DOM(y) = \Pi_{scheme(y)}(DOM(x_1) \bowtie \ldots \bowtie DOM(x_k)).$$

The meta-variables $x_1, x_2, \ldots, x_k$ of the clique become the children of y in the SD Synthesis Tree.

Step 3 Replace the clique $x_1, x_2, \ldots, x_k$ by y in the SD Graph.

Step 4 Proceed with step 1 until there are no more arcs left in the SD Graph.

The algorithm terminates, because there is only a finite number of arcs in the SD Graph (corresponding to a finite number of variables in the system description), and during each iteration, a non-empty set of arcs is removed from the graph in step (3) of the algorithm. The result of the algorithm is a SD Tree of the CSP defined by the system description. However, as the system description can consist of several unconnected portions, the result is in general a set of SD Trees rather than a single SD Tree. For simplification, in the following we assume that the result of the above algorithm is a single SD Tree. The extension to a set of several SD Trees is then straightforward.

The number of meta-variables in the SD Tree is dependent on the number of constraints in the SD graph, and thus the number of variables in the system description. In the worst case, each clique is formed by just one arc labeled with just one variable. In this case, if the number of variables in the system description is n and the number of components in the system description is c, the number of meta-variables in the SD Tree is $c + n$ (c meta-variables of the SD Graph corresponding to leaf nodes, and n meta-variables corresponding to intermediate nodes).

If the considered clique consists of only one node in the SD Graph, the corresponding node in the SD Tree has only one child. Hence, in the SD Tree resulting

from the above algorithm, the branch factor of a node can be equal to one. This occurs when the variables S considered in step (2) of the algorithm are labels of so-called *self-arcs* that refer to a single meta-variable. From the point of view of the conceptual layer, these variables correspond to variables that are not shared with other components, which means that are they are local to the component or super-component. It is possible to optimize the SD Tree by representing nodes that have a branch factor of one together with their child node within a single meta-variable. This can be accomplished by attributing a node V both with a meta-variable $V(x)$ and a meta-variable $V(x_{shared})$ with a reduced scheme comprising only the variables that are further shared with other meta-variables:

Algorithm (Generation of normalized SD Tree)

Step 1 For each node V in the SD Graph, let NS be the labels of any self-arcs of MV. Call Reduce(V,NS). Remove self-arcs of V in the SD Graph.

Step 2 Identify cliques in the SD Graph.

Step 3 Build a new meta-variable y for the clique $V_1, V_2, \ldots, V_k$ connected by arcs labeled with the set of variables S. Specify the scheme of y as the union of the schemes of the meta-variables in the clique:

$$scheme(y) = \bigcup_{i=1,\ldots,k} scheme(V(x_i))$$

Specify the domain values of x as the tuples in the join of the domains of the meta-variables $V_i(x_{shared})$ in the clique:

$$DOM(y) = DOM(V_1(x_{shared}) \bowtie \ldots \bowtie DOM(V_k(x_{shared})).$$

The nodes $V_1, V_2, \ldots, V_k$ become the children of y in the SD Tree.

Step 4 Replace the clique $V_1, V_2, \ldots, V_k$ by a node V for y in the SD Graph.

Step 5 Call procedure Reduce(V,S).

Step 6 Proceed with step 1 until there are no more arcs left in the SD Graph.

Procedure Reduce (V,NS)

Step 1 Attribute node V with a metavariable x_{shared}. Specify the scheme of x_{shared} to be the scheme of $V(x)$ without the variables NS:

$$scheme(x_{shared}) = scheme(V(x)) \setminus NS.$$

Specify the domain values of x_{shared} as the tuples in the domain of the meta-variable x, projected on the variables in $scheme(x_{shared})$:

$$DOM(x_{shared}) = \Pi_{scheme(x_{shared})}(DOM(V(x))).$$

The resulting normalized SD Tree has a branch factor greater than one. In a normalized SD Tree, for each variable v_i in the system description, there exists exactly one meta-variable $x = MV(v_i)$ such that $v_i \in scheme(x) \setminus scheme(x_{shared})$. Generating a normalized SD Tree can thus be viewed as a method to transform an arbitrary CSP corresponding to a system description to a new CSP whose constraint graph is equal to a tree.

The number of meta-variables in the normalized SD Synthesis Tree is bound by $c + n_s$, where $n_s \leq n$ is the number of shared variables in the system description. The set of shared variables can be considerably smaller than the set of all variables, as it does not involve internal variables and mode variables of the components. Figure 6.2 shows an example of a SD Tree for the pedal position sensor model.

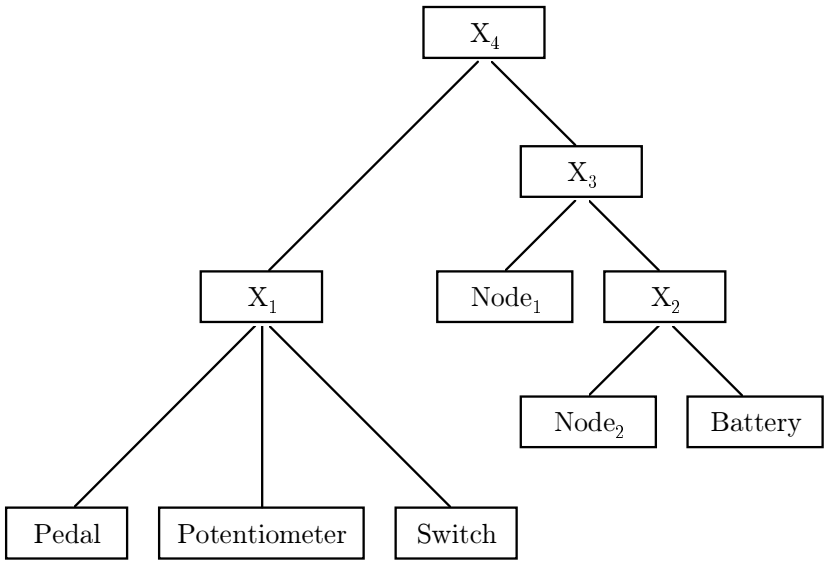

Figure 6.2: An SD Tree for the Pedal Position Sensor

6.2.5 Heuristics for SD Tree Topologies

Which meta-variables (i.e. super-components) of an SD Tree are formed is determined by the cliques in the SD Graph to be used for clustering. Thus, the topology of the SD Tree is dependent on the choice of cliques in step (2) of the algorithm. Since the complexity of subsequent steps, e.g. of minimizing the SD Tree, is dependent on the domain size of the meta-variables, the goal is to choose a topology for the synthesis tree that keeps the domain sizes of the meta-variables as small as possible. This section presents heuristics in order to choose cliques that lead to a resulting SD Tree that has small domain sizes of the meta-variables. Note that any kind of clustering strategy affects only the efficiency of the computation of the model relation R, but not its correctness. The domain size of the

meta-variables corresponds to the number of the possible solutions of the subproblem induced by the variables in the scheme of the meta-variables. In worst case, it is a combinatorial combination of the domain sizes of these variables. For a small domain size of a meta-variable, it is necessary that either

(1) the number of constraints in the subproblem is maximized, such that a large amount of combinations is removed, or

(2) the number of variables in the scheme is minimized, such that the number of possible combinations is limited from the outset.

Following the first approach amounts to finding tightly constrained subproblems in the constraint graph. A tightly constrained subproblem can be expected if the number of constraints in the cluster is maximal. Identifying such subproblems can be conceived as a graph-theoretical optimization problem, where the goal is to find a decomposition of a graph into clusters that minimize the number of edges between the clusters. However, this is a NP-hard problem (see [WF97],[WF99] and the discussion in section 6.6).

In the following, we introduce a simple heuristic for identifying subproblems forming meta-variables that is based on the second approach instead. It consists of selecting cliques in the SD Graph that have a minimum number of arcs to the remaining elements of the SD Graph. In this case, the scheme of the meta-variable involves a minimal number of variables from the system description, and thus we can hope that the domain size of the meta-variable — which is limited by the cross-product of the domains of the involved variables — is small. This optimization is repeatedly applied for each clustering step. Compared to clustering heuristics that are based on identifying tightly constrained subproblems, it can be viewed as a bottom-up heuristic. The algorithm for building the SD Tree is thus refined as follows.

Step 1a From the set of cliques, choose a clique that has a minimum number of arcs to the remaining elements of the SD Graph.

Figure 6.3 shows a SD Tree that is the result of applying this heuristic to the model of the pedal position sensor. The heuristic does not guarantee an optimal topology of the tree, because the true domain size of the meta-variables is not only determined by the number of the involved variables, but also by the resulting constraint type (i.e., its tightness). However, we found for the device models we considered in our examples that the size of meta-variables could already be kept reasonably small using the heuristics described above, meaning that it was feasible to represent the resulting domain values of the meta-variables. E.g. for the behavior model used in chapter 9, the scheme of the largest meta-variable involves less than 8 percent of the model variables, despite the presence of several feedback loops in the model. In general, the step of building the SD Tree and the subsequent step of minimizing it (see section 6.2.6) did not present severe difficulties and were possible for all of the considered examples.

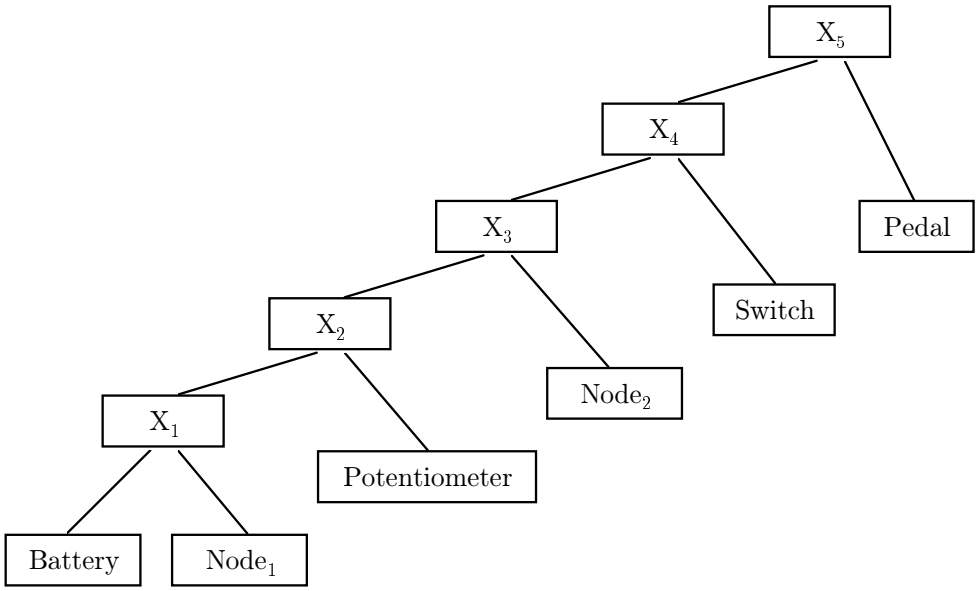

Figure 6.3: SD Tree for the Pedal Position Sensor obtained by applying the heuristic described in section 6.2.5

6.2.6 Minimizing SD Trees

A synthesis tree allows to establish consistency between the variables of the original CSP by establishing consistency among the meta-variables. Thus, computing the model relation R amounts to establishing consistency in the SD Tree.

Dechter and Pearl [DP88] observe that directional arc consistency (DAC) is sufficient for determining global consistency in a tree-structured (acyclic) CSP. DAC removes inconsistent domain values from a CSP by examining every arc exactly once (see [Tsa93]). Since the SD Tree is acyclic, establishing directional arc consistency between the meta-variables is a sufficient condition for consistency among all variables in the system description. This provides the foundation for the following algorithm to minimize a SD Tree by establishing DAC:

Algorithm (Minimization of SD Tree)

Step 1 Start with root node of the tree. Set it on the agenda.

Step 2 Choose a node from the agenda and remove it from the agenda. If the node is a leaf, then goto step 5, else determine the children of the node.

Step 3 Establish directional arc consistency between the node and its children.

Step 4 Put all the children of the node on the agenda.

Step 5 If the agenda is non-empty, proceed with step 2.

The complexity of achieving directional arc consistency in a synthesis tree is $O(mk^2)$, where m is the number of nodes (i.e. meta-variables) in the tree, and k the (maximum) number of domain values. After minimization, solutions for the original CSP can be generated from the synthesis tree in a backtrack-free manner with a complexity that is linear in the size of the tree. The basic procedure is described in [WF99]. A minimal SD Tree implicitly represents all tuples of the model relation R. It will be used as the basic representation to compute induced distinctions for a model relation. In the following, when referring to a SD Tree as a data structure, we mean a minimal, normalized SD Tree.

6.3 Basic Operations on Model Relations

The previous sections provided us with an implicit representation of the model relation R corresponding to a system description. The computation is independent of the particular task in terms of observable and target distinctions $\boldsymbol{\tau_{obs}}$ and $\boldsymbol{\tau_{targ}}$. Hence, in order to derive induced distinctions for several qualitative abstraction problems that involve the same behavior model, the corresponding SD Tree can be re-used for different combinations of observable or target partitions.

In chapter 5, we have seen that the computation of induced distinctions for a variable v_i requires to determine the observation partition $\Omega(R, \boldsymbol{\tau_{obs}})$ and the solution partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$. The remaining sections of this chapter describe how these partitions can be computed efficiently for a representation of the model relation as an SD Tree. To this end, in the following it is described how basic operations necessary to compute $\Omega(R, \boldsymbol{\tau_{obs}})$ and $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ can be carried out efficiently using a SD Tree, without the need to refer to the base model relation explicitly. The basic operations are

(1) to restrict an individual variable of the behavior model to a subset of its domain, as required e.g. to compute the elements $R_{OBS,k}$ of $\Omega(R, \boldsymbol{\tau_{obs}})$;

(2) to project the model relation onto an individual variable, as required e.g. to determine redundant domain values of this variable;

(3) to abstract the model relation by applying a domain abstraction, as required e.g. to compute the elements $R_{SOL,OBS,k}$ of $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$.

6.3.1 Projection of the Model Relation

The projection of the model relation represented by a SD Tree on an individual variable v_i can be computed by retrieving the meta-variable $MV(v_i)$ for v_i and projecting its domain on v_i:

Function Project (v_i)

Step 1 Retrieve the meta-variable $x := MV(v_i)$.

Step 2 Return $\Pi_{v_i}(DOM(x))$.

The complexity of this operation is equal to the complexity of projecting the relation $DOM(x)$ on v_i, i.e. $O(k)$ if k is the size of this relation.

6.3.2 Restriction of the Model Relation

The restriction of a variable v_i to a subset $P_{i,k}$ of its domain for a model relation represented by a SD Tree can be computed by retrieving the meta-variable $MV(v_i)$, restricting it to the domain values consistent with $P_{i,k}$, and establishing consistency with the remaining meta-variables:

Function Restrict $(v_i, P_{i,k})$

Step 1 Retrieve the meta-variable $x := MV(v_i)$.

Step 2 Restrict $DOM(x)$ to $\sigma_{v_i=P_{i,k}}(DOM(x))$.

Step 3 Return the minimized SD Tree.

The complexity of this operation is determined by the complexity for minimizing the SD Tree, which is given by $O(mk^2)$ (see section 6.2.6).

6.3.3 Abstraction of the Model Relation

Determining the relation $\boldsymbol{\tau}(R)$ for a domain abstraction $\boldsymbol{\tau}$ and a model relation R represented by a SD Tree is not accomplished by applying $\boldsymbol{\tau}$ to every meta-variable of the SD Tree. Instead, a variable can only be abstracted once it is ensured that the variable is not further shared with other meta-variables of the SD Tree. This principle leads to a bottom-up evaluation of the SD Tree. For each meta-variable x in the SD Tree, let $\boldsymbol{\tau_{ns}}$ denote a domain abstraction such that

$$\tau_{ns,i} = \begin{cases} \tau_i & v_i \in scheme(x) \setminus scheme(x_{shared}) \\ \tau_{id,i} & \text{otherwise} \end{cases}$$

The domain abstraction $\boldsymbol{\tau_{ns}}$ abstracts the variables in the scheme of x that are not shared at all or only shared with meta-variables in the subtree induced by x. Based on this, the following function returns a relation that is the abstraction of the subtree induced by node V:

Function Abstract $(V, \boldsymbol{\tau})$

Step 1 If V is a leaf, return $\boldsymbol{\tau}(DOM(V(x)))$ and exit.

Step 2 Let $V_1, \ldots, V_k$ be the children of V. Return

$$\boldsymbol{\tau_{ns}}(\text{Abstract}(V_1, \boldsymbol{\tau}) \bowtie \text{Abstract}(V_2, \boldsymbol{\tau}) \bowtie \ldots \bowtie \text{Abstract}(V_k, \boldsymbol{\tau})).$$

The relation $\boldsymbol{\tau}(R)$ is then obtained by calling the function Abstract$(V, \boldsymbol{\tau})$ with V being equal to the root of the SD Tree. The complexity of this operation is $O((k_a)^m)$, where $k_a \leq k$ is the maximal size of the abstracted domain $\boldsymbol{\tau}(DOM(x))$ for a meta-variable x.

6.4 Computing Induced Distinctions

In chapter 5, it was stated how induced distinctions for a qualitative abstraction problem QAP can be determined from the observation partition $\Omega(R, \boldsymbol{\tau_{obs}})$ and the solution partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$. The following sections describe how observation partitions and solution partitions can be computed for a model relation based on the operations described in section 6.3.

6.4.1 Determining Observation Partitions

The partition $\Omega(R, \boldsymbol{\tau_{obs}})$ consists of the subsets $R_{OBS,k}$ of R that are consistent with different external restrictions OBS_k. Assume we are given a tuple $OBS_k \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$. Then using the SD Tree representing R,

$$R_{OBS,k} := R \bowtie OBS_k$$

can be determined by restricting each variable v_i to $\Pi_{v_i}(OBS_k)$ and minimizing the SD Tree (see section 6.3.2). However, since this must be repeated for each $OBS_k \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$ in order to obtain the elements of $\Omega(R, \boldsymbol{\tau_{obs}})$, this is infeasible for large sets of possible external restrictions. Another problem is that the number of tuples OBS_k can be too large for representing the resulting set $\Omega(R, \boldsymbol{\tau_{obs}})$ by explicitly enumerating its elements. Fortunately, the implicit representation of the model relation in the form a SD Tree can also provide the basis for an implicit representation of partitions of the model relation. In the following, we present an algorithm that

(1) represents the observation partition implicitly within the SD Tree,

(2) decomposes the computation of $R_{OBS,k}$ for the individual variables v_i.

The first technique is based on the idea that the relations $R_{OBS,k}$ can be represented within the SD Tree. Given a tuple OBS_k, instead of removing the domain values of each meta-variable that are inconsistent with OBS_k and keeping the domain values that are consistent, we can represent the consistency or inconsistency of domain values with OBS_k as a partition for the meta-variables.

Definition 50 (Domain Partition of Meta-Variable) *A domain partition of a meta-variable x is partition of the set $DOM(x)$.*

By labeling the obtained partition elements of meta-variables as either consistent or inconsistent with the given external restriction, we can keep track of the information which meta-variable partition element belongs to which observation partition element $R_{OBS,k}$. Note that the domain of a meta-variable x corresponds to a set of tuples, hence a partition element of a meta-variable corresponds to a relation over the variables in $scheme(x)$.

The implicit representation of $\Omega(R, \boldsymbol{\tau_{obs}})$ through partitions of meta-variables in the SD Tree greatly reduces the overall number of partition elements that are

needed to represent the observation partition. While the observation partition can consist of as many partition elements as tuples in R, there are maximal $m \cdot k$ partition elements in a SD Tree, where is m the number of meta-variables in the SD Tree, and k the maximal domain size of a meta-variable.

The second technique is based on the idea that it is not necessary to iterate over all $OBS_k \in \boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$. Instead, the consistency or inconsistency with external restrictions can be determined by successively considering the observable partition elements $P_{i,k} \in \pi_{obs,i}$ of the individual variables v_i. Given a qualitative abstraction problem QAP with the model relation R represented as a SD Tree and observable distinction $\boldsymbol{\tau_{obs}}$, the following algorithm determines the observation partition:

Algorithm (Computation of Observation Partition)

Step 1 For each meta-variable x, set π_Ω equal to the trivial partition $\{DOM(x)\}$.

Step 2 Choose a variable v_i and an observable partition element $P_{i,k} \in \pi_{obs,i}$.

Step 3 Determine the restriction of R to $v_i = P_{i,k}$. For each meta-variable x, this yields a partition $\pi_x = \{P_{cons}, P_{incons}\}$ where $P_{cons} \subseteq DOM(x)$ denotes domain values that are consistent and $P_{incons} \subseteq DOM(x)$ domain values that are inconsistent with the restriction.

Step 4 For each meta-variable x, set $\pi_\Omega := MERGE(\pi_\Omega, \pi_x)$. For each partition element of π_Ω that is a subset of P_{cons}, label it as consistent with the observation $v_i = P_{i,k}$.

Step 5 Proceed with step 2 until all variables and all observable partition elements have been considered.

The result is an implicit representation of the observation partition through partitions π_Ω of the meta-variables in the SD Tree. Each partition element of a meta-variable has a label representing the partition elements $R_{OBS,k}$ it belongs to.

Theorem 4 (Computation of Observation Partition) *For a model relation R given as an SD Tree, the partition $\Omega(R, \boldsymbol{\tau_{obs}})$ can be computed in time polynomial in n, o, m and k, where n is the number of variables, o the maximal number of elements in the domain partitions $\pi_{obs,i}$, m the number of meta-variables in the SD Tree, and k the maximal domain size of a meta-variable.*

Proof. In the algorithm above, maximally $n \cdot o$ restrictions of R have to be considered. Each such restriction can be computed in $O(mk^2)$. In step 3 of the algorithm, maximally k partition elements of a meta-variable can occur. □

6.4.2 Determining Solution Partitions

The elements of the partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ are formed by the elements $R_{OBS,k}$ of the observation partition that are consistent with the same tuples of the set $\boldsymbol{\tau_{targ}}(DOM(\boldsymbol{v}))$ (see section 5.7.1). In order to obtain the solution partition, we have to determine, for each element $R_{OBS,k}$ of the observation partition, the tuples of the set $\boldsymbol{\tau_{targ}}(DOM(\boldsymbol{v}))$ that $R_{OBS,k}$ is consistent with. Assume we are given a tuple $SOL_j \in \boldsymbol{\tau_{targ}}(DOM(\boldsymbol{v}))$. Then the $R_{OBS,k}$ that are consistent with SOL_j can be determined by restricting each variable v_i to $\Pi_{v_i}(SOL_j)$ and minimizing the SD Tree. Hence, in principle, the solution partition can be determined by iterating over the tuples SOL_j and identifying the elements in the observation partition that are consistent with different sets of SOL_j.

We can apply the same principles as outlined in section 6.4.1 in order to represent the solution partition implicitly within the SD Tree and to decompose the computation of the $R_{SOL,OBS,k}$ for individual variables v_i. Again, the consistency or inconsistency of parts of the model relation with a specific SOL_j is reflected in the form of partition elements for the meta-variables.

Given an observation partition $\Omega(R, \boldsymbol{\tau_{obs}})$ represented as partitions of meta-variables in a SD Tree and a target distinction $\boldsymbol{\tau_{targ}}$, the following algorithm determines the solution partition. Here, $DOM_\Omega(x)$ denotes the elements of the partition π_Ω of x:

Algorithm (Computation of Solution Partition)

Step 1 For each meta-variable x, set π_Σ equal to trivial partition $\{DOM_\Omega(x)\}$.

Step 2 Choose a variable v_i and a target partition element $P_{i,k} \in \pi_{targ,i}$.

Step 3 Determine the restriction of R to $v_i = P_{i,k}$. For each meta-variable x, this yields a partition $\pi_x = \{P_{cons}, P_{incons}\}$ of $DOM_\Omega(x)$ where $P_{cons} \subseteq DOM_\Omega(x)$ denotes elements that are consistent and $P_{incons} \subseteq DOM_\Omega(x)$ elements that are inconsistent with the restriction.

Step 4 For each meta-variable MV, set $\pi_\Sigma := MERGE(\pi_\Sigma, \pi_x)$. For each partition element of π_Σ that is a subset of P_{cons}, label it as consistent with the solution $v_i = P_{i,k}$.

Step 5 Proceed with step 2 until all variables and all target partition elements have been considered.

The result is an implicit representation of the solution partition through partitions π_Σ of the meta-variables in the SD Tree. Each partition element of π_Σ is labeled with tuples $SOL \in \boldsymbol{\tau_{targ}}(DOM(\boldsymbol{v}))$.

Theorem 5 (Computation of Solution Partition) *For a model relation R given as an SD Tree, the partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ can be computed in time polynomial in n,s, m and k, where n is the number of variables, s the maximal number of elements in the domain partitions $\pi_{targ,i}$, m the number of meta-variables in the SD Tree, and k the maximal domain size of a meta-variable.*

The proof is analog to the proof of theorem 4.

Example 17 (PPS Model) *Consider the qualitative abstraction problem that consists of the pedal position sensor model described in example 1, the observable distinctions described in example 5, and the target distinction described in example 6. For the SD Tree of the pedal position sensor model shown in figure 6.3, node X_5 is associated with a meta-variable x that has a single variable in its scheme:*

$$scheme(x) = Pedal.position$$

Its domain consists of six values:

$$val_1 = 0\%,\ val_2 = 20\%,\ val_3 = 40\%,\ val_4 = 60\%,\ val_5 = 80\%,\ val_6 = 100\%.$$

The partition π_Ω for x consists of six partition elements that are consistent with different sets of observations:

- $P_{\Omega,1} = \{val_1\}$ *with the observation* $v_{pot} = [0V, 2V)$, $v_{switch} = [0V, 2V)$,
- $P_{\Omega,2} = \{val_2\}$ *with the observations* $v_{pot} = [0V, 2V)$, $v_{switch} = [0V, 2V)$ *and* $v_{pot} = [2V, 4V)$, $v_{switch} = [0V, 2V)$,
- $P_{\Omega,3} = \{val_3\}$ *with the observations* $v_{pot} = [2V, 4V)$, $v_{switch} = [0V, 2V)$ *and* $v_{pot} = [4V, 6V)$, $v_{switch} = [0V, 2V)$,
- $P_{\Omega,4} = \{val_4\}$ *with the observations* $v_{pot} = [4V, 6V)$, $v_{switch} = [8V, 10V)$ *and* $v_{pot} = [6V, 8V)$, $v_{switch} = [8V, 10V)$,
- $P_{\Omega,5} = \{val_5\}$ *with the observations* $v_{pot} = [6V, 8V)$, $v_{switch} = [8V, 10V)$ *and* $v_{pot} = [8V, 10V)$, $v_{switch} = [8V, 10V)$,
- $P_{\Omega,6} = \{val_6\}$ *with the observation* $v_{pot} = [8V, 10V)$, $v_{switch} = [8V, 10V)$.

The partition π_Σ for x consists of two partition elements:

- $P_{\Sigma,1} = \{P_{\Omega,1}, P_{\Omega,2}, P_{\Omega,3}\}$ *consistent with the solution* $v_{switch} = [0V, 2V)$,
- $P_{\Sigma,2} = \{P_{\Omega,4}, P_{\Omega,5}, P_{\Omega,6}\}$ *consistent with the solution* $v_{switch} = [8V, 10V)$.

□

6.4.3 Verifying Properties of Qualitative Abstraction Problems

For the sections above, we have assumed that for a given qualitative abstraction problem $QAP = (R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$, the preconditions necessary for applying theorems 2 and 3 in chapter 5 hold. In this section, we are concerned with the problem of how to determine whether a given QAP fulfills these preconditions. The preconditions to fulfill are minimality, observability, and consistency of QAP. The latter is only necessary in the case where a complete solution is requested. Using a representation of R as a SD Tree, these preconditions can be checked as follows:

(1) checking minimality of QAP can be done after minimization of the SD Tree by computing the projection of R on each variable v_i in oder to determine redundant domain values (alternatively, minimality can be *established* by removing the redundant domain values from the domains);

(2) checking observability of QAP corresponds to checking whether for each target partition element $P_{i,k}$ of a variable v_i, there exists an element of $DOM_\Omega(MV(v_i))$ that is consistent with $P_{i,k}$ and inconsistent with all other target partition elements (this can be verified in step (4) of the algorithm in section 6.4.2);

(3) checking the consistency of QAP is equivalent to checking if the abstraction of the model relation $\boldsymbol{\tau_{obs}}(R)$ is equal to the space of external restrictions $\boldsymbol{\tau_{obs}}(DOM(\boldsymbol{v}))$.

Figure 6.4 summarizes the computational steps necessary for solving a qualitative abstraction problem.

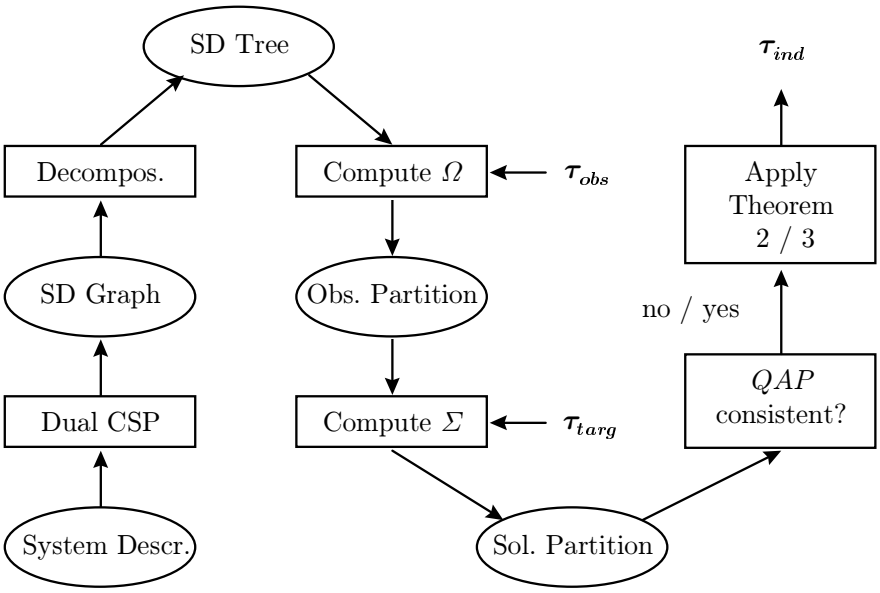

Figure 6.4: Computational Steps for Deriving Induced Distinctions

6.5 Transforming System Descriptions

The elements $R_{SOL,OBS,k}$ of the solution partition can finally be used to compute the induced distinctions. According to theorem 2 in chapter 5, an abstraction $\pi'_{ind,i}$ of $\pi_{ind,i}$ can be derived by determining the projection of each $\boldsymbol{\tau_{obs}}(R_{SOL,OBS,k})$ on the variable v_i. Theorem 3 in chapter 5 states that $\pi_{ind,i}$ is

obtained by determining the interchangeable values for variable v_i with respect to each $\boldsymbol{\tau_{obs}}(R_{SOL,OBS,k})$. As theorem 1 in chapter 5 states, the result has to be merged with the target distinction $\boldsymbol{\tau_{targ}}$. The qualitative abstraction $\boldsymbol{\tau_{ind}}(R)$ of the behavior model can finally be obtained by applying the corresponding domain abstraction $\boldsymbol{\tau_{ind}}$ to R (see section 6.3.3).

The induced distinctions $\boldsymbol{\tau_{ind}}$ and the transformed behavior model $\boldsymbol{\tau_{ind}}(R)$ constitute in principle the solution to the problem of automated qualitative abstraction. However, the employed model-based problem solving framework (see section 7.1.1) is currently not able to process the transformed behavior model $\boldsymbol{\tau_{ind}}(R)$ directly due to the involved additional behavior constituents (i.e., super-components). Instead, the result has to be mapped back to the original model fragments of the system description. If $\boldsymbol{\tau_{ind}}$ has to be mapped back to the behavior model fragments of the system description, a problem arises since $\boldsymbol{\tau_{ind}}$ cannot be simply applied to each of the model fragments. This is illustrated by the following example.

Example 18 (Cascaded Equality) *Let* $\boldsymbol{v} = (v_1, v_2, v_3)$, $DOM(v_i) = \{0, 1, 2\}$. *Let* R *express the behavior* $v_1 = v_2 = v_3$, *i.e.*

$$R(\boldsymbol{v}) = \{(0,0,0), (1,1,1), (2,2,2)\}.$$

Assume that the only non-trivial observable partition is given for variable v_1

$$\pi_{obs,1} = \{\{0\}, \{1\}, \{2\}\}$$

and that the only non-trivial target distinction is to determine whether v_3 *is zero or not:*

$$\pi_{targ,3} = \{\{0\}, \{1,2\}\}.$$

Then the induced distinctions are

$$\pi_{ind,1} = \{\{0\}, \{1,2\}\}, \pi_{ind,2} = \{\{0,1,2\}\}, \pi_{ind,3} = \{\{0\}, \{1,2\}\}.$$

Let the corresponding domains be $DOM_{ind,i}$. *Assume that* $R(\boldsymbol{v})$ *has been composed of two relations* R_1 *and* R_2

$$R_1(v_1, v_2) = \{(0,0), (1,1), (2,2)\},$$
$$R_2(v_2, v_3) = \{(0,0), (1,1), (2,2)\}.$$

Then applying $\boldsymbol{\tau_{ind}}$ *to the original relations* R_1 *and* R_2 *yields*

$$R_1'(v_1, v_2) = \{(\{0\}, \{0,1,2\}), (\{1,2\}, \{0,1,2\})\}$$
$$R_2'(v_2, v_3) = \{(\{0,1,2\}, \{0\}), (\{0,1,2\}, \{1,2\})\}$$

and composing the relation R *again results in*

$$R_1' \bowtie R_2' = DOM_{ind,1} \times DOM_{ind,1} \times DOM_{ind,1},$$

i.e. the composed relation contains no restriction. □

This illustrates that if the relation $\tau_{ind}(R)$ is to be decomposed into the original behavior model fragments, possibly more distinctions are needed than provided by the induced distinctions. The problem is related to representing an n-ary relation as a set of relations with arity smaller than n. [RPD90] shows that it is in principle possible to break down relations with higher arity into relations with smaller arity without altering the set of solutions.

Below, we describe an approach to this problem that is based on the algorithms already developed in this chapter. The idea is that all variables in the model must have enough distinctions to represent any reduction of their domain resulting from an external restriction. Accordingly, the approach to deriving the additional distinctions for a compositional model is as follows:

Algorithm (Additional Distinctions for Compositional Model)

Step 1 Let $\tau_{obs}{}'$ be the abstraction of τ_{obs} derived by theorems 2 and 3.

Step 2 Compute the observable partition $\Omega(R, \tau_{obs}{}')$.

Step 3 Set the solution partition equal to the identical partition, i.e. each element of the observation partition constitutes a partition element of the solution partition that needs to be distinguished.

Step 4 Compute the induced distinctions for the solution partition.

For the resulting domain abstraction, the transformed behavior model fragments contain sufficient distinctions in order to obtain the same restrictions as the transformed composed model in the case of orthogonal solutions (see section 3.6.1). For the example above, the additional partition $\{\{0\}, \{1, 2\}\}$ for variable v_2 is derived.

6.6 Discussion

The methods described in this chapter are based on the SD Tree as a central data structure that "compiles" the model relation R into a an implicit representation in order to avoid combinatorial explosion. It has been shown how such an implicit representation can be used as a basis to efficiently derive task-dependent qualitative abstractions of R. It is therefore worth to discuss in more detail under which preconditions such a representation is feasible.

The SD Tree data structure is adapted from the method described in [WF99], which is an extension of the tree clustering method described in [DP88]. Weigel and Faltings already note that the synthesis tree is a data structure that is especially well-suited for repeated "queries", because in this situation, the necessary effort for compilation pays off. The approach presented in this chapter employs this idea. However, there are many differences of the approach described in this chapter to the work described in [WF99]. Weigel and Faltings represent the domains of meta-variables explicitly by enumerating the respective tuples. They

use local (neighborhood) interchangeability to get a more compact representation of the domains of meta-variables. This causes extra effort for computing interchangeable values, as meta-values must be updated during construction of the synthesis tree. We use an OBDD representation for the domains of meta-variables instead. Our experiments indicate that this leads to a far more efficient representation than representing the domains by enumerating their elements. Weigel and Faltings are concerned only with the efficient representation of solutions to constraint satisfaction problems, and not with applying this representation within the context of model abstraction.

The computational complexity of the operations on the SD Tree, hence the complexity of all of the algorithms described in this chapter, is determined by the factor k that denotes the maximal domain size of a meta-variable. Section 6.2.5 described a heuristic for optimizing the topology of the resulting SD Tree with the goal to minimize k. Weigel and Faltings ([WF99]) present a top-down heuristic for identifying subproblems in a CSP that is based on a decomposition method called recursive spectral bisection. It applies a median cut procedure that determines a partitioning of the variables in the constraint graph such that the number of inter-partition edges is minimized. The results are used for a bottom-up decomposition algorithm that heuristically finds cliques of variables to be clustered into meta-variables. This heuristic is related to the maximum degree variable ordering heuristic in backtrack search (see [Tsa93]), where the node to be instantiated next is the one with the most constraints on the already instantiated variables.

[GLS00] provides a comparison of different CSP decomposition methods based on the classes of problems that are tractable, and presents a method called hypertree decomposition that is optimal in the sense that its class subsumes those of other known decomposition methods. It is also shown that for the special case of binary CSPs (corresponding to a SD Graph), hypertree decomposition is essentially equivalent to the tree clustering method ([DP88]).

Using the SD Tree, the computational complexity of deriving task-dependent qualitative abstraction can be bound to structural properties of the system description. This is the basis to identify tractable subsets of qualitative abstraction problems. The principle is that if the maximal size of meta-variables is bound, then the complexity of deriving task-dependent qualitative abstractions is polynomial (see also the theorems 4 and 5).

Resistive networks ([Mau98], [Ran98]) are a special class of devices that can be modeled by a class of component types describing the generation, transportation and consumption of energy. [Ran98] observes that for clustering such component types, the resulting super-components have the same relation types as the original component relations. This means that the meta-variables correspond again to components. [Ran98] calls this property the closure property of component types describing resistive networks. This means that for behavior models describing resistive networks, the maximal size k of meta-variables is constant and equal to the size of the original component relations. Hence, resistive networks correspond to a class of problems where task-dependent qualitative abstraction is tractable.

The clustering step performed in building synthesis trees is related to the compilation method described in [dK92]. In this paper, de Kleer proposes a compilation step for structures in the system description called "modules" (which can consist of a subset of components of the system). The compilation derives the prime implicates of the respective constraints. Hence, for a given input-output structure, boolean constraint propagation is sufficient to determine the result. The problem with this approach is that the compiled device grows exponentially in the size of inputs and outputs. De Kleer observes e.g. that for an example involving a 20-bit multiplier, the approach becomes infeasible. De Kleer's approach is equivalent to deriving the relation R explicitly for a subset of variables corresponding to input and output terminals of the module. The method presented in this chapter also determines the relation R, but in contrast to de Kleer's algorithm, it provides no explicit representation for it.

The chapter was not concerned with the questions of what aspects of the application problems outlined in chapter 2 can be addressed by the algorithms, and how they fit into the general framework for model-based problem solving outlined in chapter 3. This will be the topic of the next chapter.

6.7 Summary

This chapter put the computation of task-dependent qualitative model abstractions to work by developing efficient algorithms to compute induced distinctions based on propagation of constraints. The algorithms use the SD Tree as a data structure that represents the behavior model relation implicitly and can exploit problem-specific features in order to avoid combinatorial explosion.

The complexity of deriving solutions to a qualitative abstraction problem is thus bound to the problem-specific structure of the device model. This is the basis to scale up to real-world problems (an example will be given in chapter 9). The implementation of these algorithms and their integration into a general framework for model-based problem solving will be the topic of the next chapter.

Chapter 7

A Prototypic System for Task-dependent Qualitative Model Abstraction

This chapter describes how computer-supported qualitative model abstraction can be embedded into — and extends — an existing framework for model-based prediction and diagnosis.

The first section of this chapter outlines an existing model-based framework and its basic software components and interfaces. Then, an enhanced framework is proposed that incorporates three additional prototypic software components that implement the methods developed in chapters 5 and 6.

One software component generates transformed system descriptions. It receives a base system description and a domain mapping for the variables, producing as output an abstracted system description that is the result of applying the domain mapping to the base system description.

A second software component computes task-dependent qualitative domain abstractions. Given a qualitative abstraction problem, i.e. a system description and a description of the task in terms of target and observable distinctions, it computes a domain mapping corresponding to induced distinctions.

The third software component applies qualitative domain abstractions to real-valued, time-varying data. Based on the domain mapping underlying the transformed model, this component outputs observations (i.e., external restrictions) that are at the level of granularity of the transformed model. The resulting qualitative observations can be used to perform model-based prediction and diagnosis for the transformed model.

The second part of the chapter presents a number of principled examples that illustrate how the enhanced framework can be used to support automated modeling, and in particular the re-use of numerical models in the context of model-based systems.

7.1 Overview of Components and Interfaces

This section provides an overview of a prototypic system that implements automated, task-dependent qualitative model abstraction. The prototype builds on components of an existing model-based reasoning framework that is described in the following section.

7.1.1 Components of the Raz'r Framework

Raz'r (see `http://www.occm.de`) is a commercial framework for model-based behavior prediction and diagnosis. Among others, it contains the following components:

- a *Constraint Type Editor* that allows to define finite domains and constraint types over these domains;
- a *Development System* that allows to define component types and device structures in order to compose system descriptions;
- a *Run-time System* that implements consistency-based diagnosis (GDE, see section 3.6.2) for a system description and a given set of observations.

Constraint types, component types and device structures are stored persistently in a model library. The constraint system underlying Raz'r's constraint network generation and reasoning represents constraints as ordered binary decision diagrams (OBDDs) (see [Bry92]). Part of the constraint system is the constraint compiler, which allows to perform basic operations on constraints, such as join or projection (see section 3.2.2). Figure 7.1 depicts the basic components of the Raz'r framework.

7.1.2 Interfaces of the Raz'r Framework

The following types of files are exchanged between the components (all files are given in *Extensible Markup Language* format (XML, see [XML00])):

`Constraint Type Library.cll` defines domains and constraint types that can be used for building component types in the development system.

`System Description.xml` defines a behavior model of a device in terms of a structural description and component types (see section 3.4).

`Observation Description.xml` contains observations (i.e. external restrictions for variables in a system description) attributed with time points.

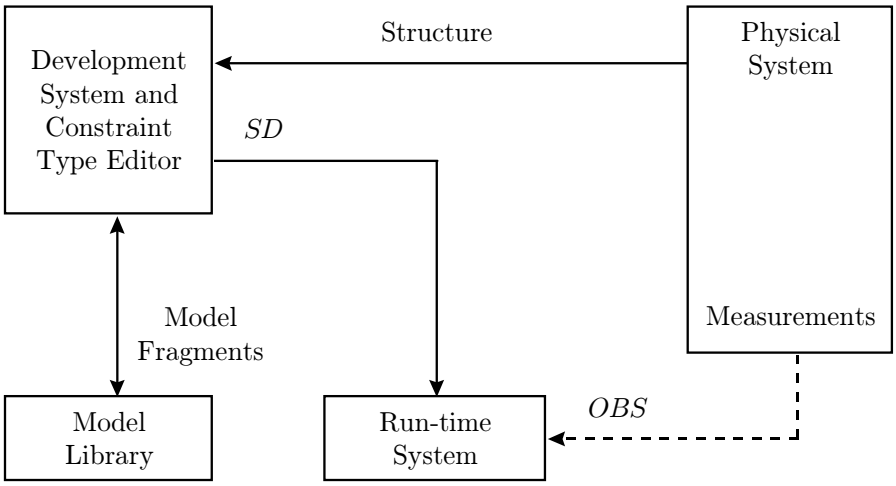

Figure 7.1: Basic components of the Raz'r framework

7.1.3 Components of the Enhanced Raz'r Framework

The ideas outlined in the previous chapters lead to the implementation of a prototypic system termed AQUA (Automated Qualitative Abstraction). Its name alludes to the vision of supporting a free "flow" between tasks requiring different granularity of knowledge. In addition to the basic Raz'r components outlined in section 7.1.1, it consists of the following three components:

- a component for the *Computation of Induced Distinctions* that determines task-dependent qualitative abstractions based on the algorithms described in chapter 6;

- a *System Description Generator* that applies a domain abstraction to a system description in order to transform the system description (in particular, the involved constraint types) to the granularity of the derived qualitative values;

- a *Signal Transformation Module* that generates qualitative observations based on real-valued, time-varying data.

Figure 7.2 depicts the components of the enhanced framework. Similar to the original Raz'r components, the components of AQUA have been implemented in Microsoft Visual Basic 5.0 under Windows NT 4.0, using Microsoft's Component Object Model (COM) technology to enable a modular design. In the following, these additional components and their interfaces will be described in more detail.

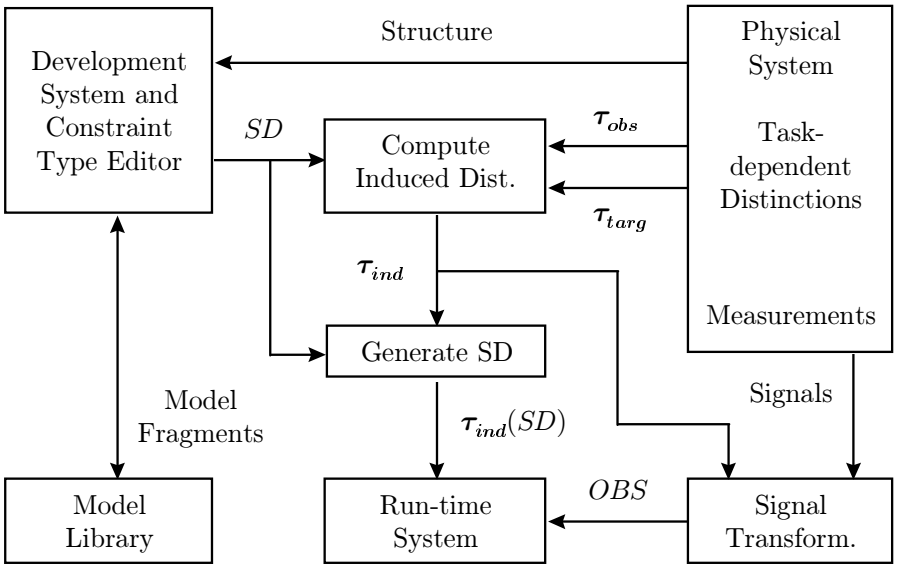

Figure 7.2: Software components of AQUA

7.1.4 Interfaces of the Enhanced Raz'r Framework

In addition to the file types present in the original Raz'r framework (section 7.1.2), AQUA uses the following file types (except for the measurement files, all files are again given in XML format):

`Partition Description.xml` defines either a target partition, observable partition, or induced partition. For example, the following partition description specifies a domain partition for v_1 that separates the values *high* and *low* from the value *medium*:

```
<Partition VARIABLE_NAME="v1">
   <PartitionElement>
      <Value VALUE="high"/>
      <Value VALUE="low"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="medium"/>
   </PartitionElement>
</Partition>
```

`Domain Mapping.xml` defines mappings from a base domain to an abstracted domain. If the base domain is finite, domain values of the abstracted domain are specified by enumerating the respective domain values of the base domain, just as for a partition description. If the base domain is equal to the real numbers, the domain values of the abstracted domain are specified

by a set of extended intervals. A set of extended intervals is any finite set of open, half-open or closed intervals over the domain of real numbers extended by $-\infty$ and ∞. For example, the domain mapping

$$v_1 = \begin{cases} medium & : \quad v_1 = 0 \\ high\_or\_low & : \quad v_1 \in (-\infty, 0) \cup (0, \infty) \end{cases}$$

can be specified as

```
<Domain NAME="Dom"
 BASEDOMAIN_NAME="Real Numbers">
   <DomainValues>
      <DomainValue VALUE="medium">
         <PartitionElement>
            <Interval LEFT_VALUE="0" LEFT_TYPE="closed"
             RIGHT_VALUE="0" RIGHT_TYPE="closed"/>
         </PartitionElement>
      </DomainValue>
      <DomainValue VALUE="high_or_low">
         <PartitionElement>
            <Interval LEFT_VALUE="MINF" LEFT_TYPE="open"
             RIGHT_VALUE="0" RIGHT_TYPE="open"/>
            <Interval LEFT_VALUE="0" LEFT_TYPE="open"
             RIGHT_VALUE="INF" RIGHT_TYPE="open"/>
         </PartitionElement>
      </DomainValue>
   </DomainValues>
</Domain>
```

`Variable Mapping.xml` defines a mapping between variables of a system description and domain mappings. The base domain of the domain mapping must be equal to the domain of the variable as specified in the system description.

`Parameter Description.xml` defines values for parameters in a system description and additional (component-specific) constraint types, such as characteristic maps. A parameter description provides a means to further specialize component types, e.g. a valve or an engine, for a specific device. Parameters and constraint types in a parameter description can be specified based on the domain of real numbers, and therefore they can be adopted more or less directly from numerical models or design specifications.

`Signal Transformation Description.xml` defines observations by specifying a mapping between real-valued signals contained in a raw data file (see below) and variables in a system description. The value for a variable at a point in time is obtained by applying the corresponding domain mapping from real numbers to its qualitative domain.

`Measurements.[xml|dat]` contains numerical, time-varying measurements obtained e.g. from a vehicle. The `.dat` format is a proprietary file format of VS100, a measurement acquisition tool that is of wide-spread use in the automotive industry (see also chapter 8). A single measurement file can consist of several signals that have been recorded simultaneously.

7.1.5 Prototype for Generation of System Descriptions

This software component generates transformed system descriptions, i.e. device models. On the one hand, a device model can be abstracted by applying domain mappings for the variables occurring in the model. The base domain of a domain mapping can be either the set of real numbers or a finite domain. On the other hand, the software component can be used to further specialize device models by applying parameter descriptions.

The software component consists of two parts. The first part is a file `Real Numbers Constraint Type Library.cll`, which contains pre-defined unary, binary and ternary constraint types such as $+$, $-$, $*$, $/$, $=$, $<$, or $\leq$ over a base domain with the key word `Real Numbers`. These constraint types can be used within the Development System and mixed with constraint types over other (finite) domains in order to define component types and system descriptions. The real-valued constraints have no meaning for the original Raz'r constraint system, and are basically treated as empty constraints over empty domains.

The second part is a tool that abstracts and specializes system descriptions, in particular the constraint types contained in a system description. The input is a system description, one or more domain mappings, one or more variable mappings, and possibly one or more parameter descriptions. The output is a modified system description that is the result of restricting parameters and component-specific constraint types through the parameter description, and abstracting the domains of the system variables by the respective domain mappings. To this end, the constraint types the variables are involved in have to be transformed and the domains of the variables have to be replaced by the new domains. In the case of a real-valued parameter or constraint type, the system description generator — based on recognizing the key word `Real Numbers` — generates the corresponding finite relation from the definition of domain values as sets of extended intervals.

The system description generator features a graphical user interface in order to select input and output files and to inform the user about intermediate results, such as the size of the generated constraints. It uses Raz'r's interfaces to read, manipulate, and write system descriptions. Figure 7.3 shows a screenshot of the system description generator.

7.1.6 Prototype for Computation of Induced Distinctions

This software component is the central part of AQUA. It computes task-dependent domain abstractions based on the algorithms described in chapter 6. The inputs are a system description and partition descriptions for observable and target dis-

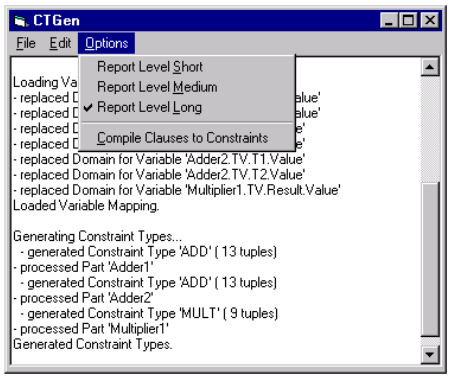

Figure 7.3: Screenshot of the System Description Generator (generating a system description for the pedal position sensor)

tinctions. If no observable partition is specified, the finest possible granularity will be assumed. The task-dependent observable or target distinctions can be either read in as files, or interactively defined by the user through a built-in graphical Partition Editor (see figure 7.5).

The software component builds an SD Tree for the model relation and uses it, as described in chapter 6, to determine observation and solution partitions. For each domain, domain values that are redundant will be put into a separate partition element (alternatively, they could be removed from the domain). The user can then choose between applying the incomplete algorithm (section 5.7.2) or the complete algorithm (section 5.7.3) to derive induced distinctions.

As an output, first, a partition description defining the induced distinctions can be generated. The resulting partition description can also be converted to HTML format in order to inspect the results using a standard web browser. Second, a domain mapping defining the mapping from the base domains to the abstracted domains can be generated. Third, the domain mapping can also be applied to the system description. Unlike the system description generator, however, the generated transformed system description does not retain the original structure of constraint types in a behavior description, but instead it uses one constraint per component (which corresponds to the dual constraint graph representation). Variables in the model that are found to have no induced distinction after computation (i.e. whose domain consists of one element only) can optionally be eliminated from the model. If all variables of a component have been eliminated, this may eventually lead to the elimination of a component from the model.

The component for the Computation of Induced Distinctions has a full-featured graphical user interface that displays a graphical representation of the system structure and intermediate results of the computation. In particular, the degree of granularity of the transformed domains, relative to their base domains, will be depicted. The software component and its implementation are described in

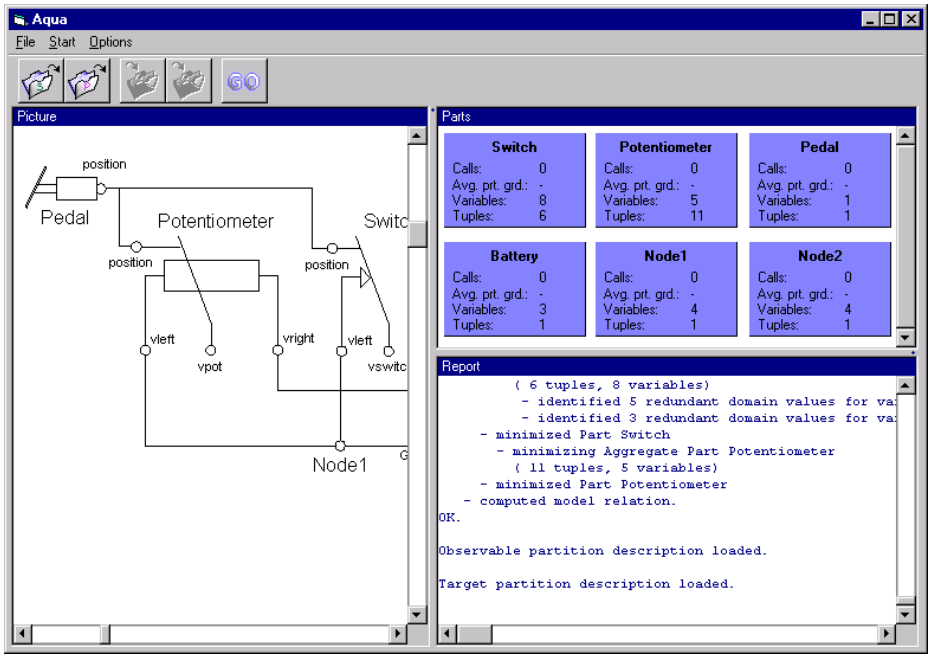

Figure 7.4: Screenshot of Computation of Induced Distinctions (deriving a task-dependent qualitative abstraction of the pedal position sensor model)

more detail in [Kut00]. Figure 7.4 shows a screenshot of the prototype for the computation of induced distinctions.

7.1.7 Prototype for Signal Transformation

This software component generates qualitative observations from real-valued, time-varying data, e.g. measurements obtained from a vehicle. The input is a measurement file specifying numerical data, a domain mapping from the real numbers to domains occuring in the system description (as generated by the component described in the previous section), and a signal transformation description.

The signal transformation component applies, at each time point, the domain mapping to the signals to obtain a qualitative abstraction of the values. Only qualitative states that are different from the previous one will be considered in the resulting output. Thus, the amount of data is often greatly reduced compared to the original numerical data (see also the examples in section 7.2 and chapter 9). In addition, the signal transformation description allows to perform basic signal preprocessing, such as specifying offsets for signals, combining several signals to obtain derived ones (e.g., calculating the difference between a pressure signal and the atmospheric pressure signal, or calculating the injection mass per time unit from the injection mass per stroke and the engine speed), and applying simple filtering methods such as smoothing.

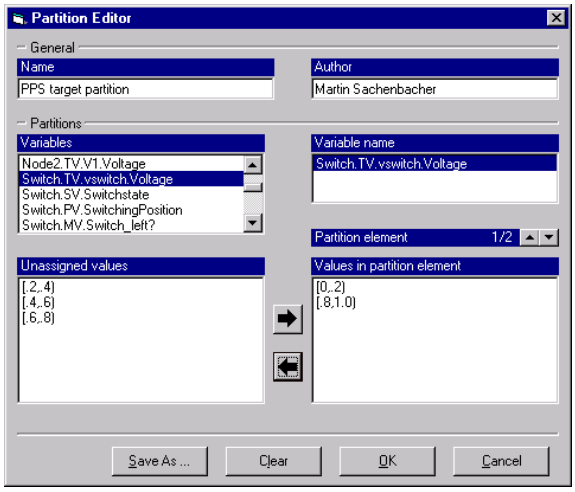

Figure 7.5: Screenshot of the Partition Editor (defining target distinctions for the pedal position sensor)

The signal transformation component supports the proprietary file format of VS100, a standard tool-set of the automotive industry that is used for acquiring and processing control unit data. Hence, it is the basis for utilizing the methods within the application domain of automotive systems. Besides that, the signal transformation component supports also an XML file format for specifying numerical measurements. It is therefore possible to perform, if necessary, more complex signal preprocessing outside this software component (e.g., using tools like Matlab) and to feed the results back into the signal transformation component. The output is an observation description that contains observations at the level of granularity of the system description. This result can be used within Raz'r to perform model-based prediction and diagnosis. Figure 7.6 shows a screenshot of the prototype for signal transformation.

7.2 Principled Use for Modeling and Building Model-based Systems

With AQUA, our prototypic system for task-dependent qualitative model abstraction, several tasks can be supported in the context of model-based problem solving that up to now essentially had to be carried out manually. The following section presents several examples, all taken from the automotive domain, that illustrate principled applications of AQUA to support modeling and automated reasoning about physical systems. Outputs shown in typewriter font have been directly generated using AQUA. All results in this section have been obtained using the incomplete algorithm (section 5.7.2). A larger example that illustrates more of AQUA's capabilities and makes use of the complete algorithm will be

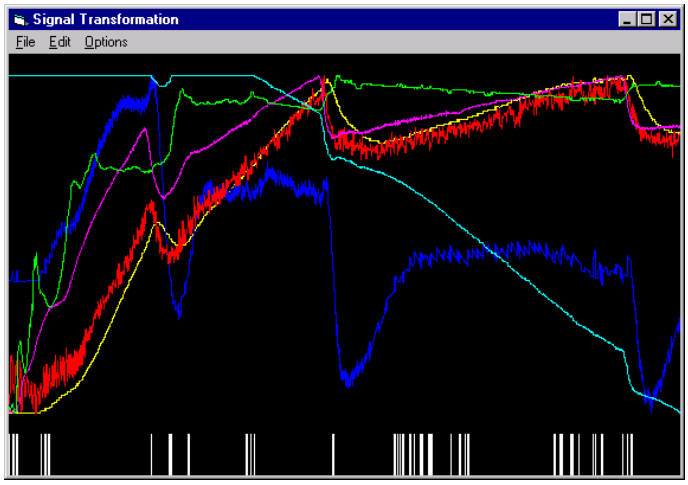

Figure 7.6: Screenshot of Signal Transformation Component (processing measurements for the Volvo Demonstrator Car)

presented in chapter 8.

7.2.1 Determining Significant Distinctions

The most obvious use of automated domain abstraction is to determine "tailor-made" distinctions for the magnitudes of variables in a model. This is illustrated by the pedal position sensor example that has been introduced in chapter 1.

Pedal Position Sensor Example

Consider again the device shown in figure 1.1. As noted in section 1.1.1, we would like to find a suitable domain for the voltage of v_{pot} that allows us to relate the value of v_{pot} with the value of v_{switch}, as required to perform tasks such as on-board diagnosis (e.g., cross-checking the signals for plausibility). A generic domain abstraction that distinguishes only between voltage *gnd*, *between* and *batt* is not adequate for this task. The problem is that the distinctions in the domain of v_{pot} must correspond to the switch-over point of the switch, hence they cannot be anticipated in a generic model of the potentiometer. In the following, we show how we can use the approach of task-dependent automated qualitative modeling and the software components outlined in this chapter to automatically derive the required granularity of v_{pot}.

Like in example 1, the domains in the base model SD_{base} are

$$DOM = \{[0V, 2V), [2V, 4V), [4V, 6V), [6V, 8V), [8V, 10V)\}$$

for variables involving voltage and

$$DOM = \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$$

for variables involving position.

The observable distinctions express the fact that the control unit can observe the signal from the potentiometer and the signal from the switch:

$$\pi_{obs,v_{pot}} = \{\{[0V, 2V)\}, \{[2V, 4V)\}, \{[4V, 6V)\}, \{[6V, 8V)\}, \{[8V, 10V)\}\},$$
$$\pi_{obs,v_{switch}} = \{\{[0V, 2V)\}, \{[2V, 4V)\}, \{[4V, 6V)\}, \{[6V, 8V)\}, \{[8V, 10V)\}\}.$$

For the parameters in the model, i.e. the nominal voltage *gnd* and *batt* of the battery and the switch-over parameter $pos_{switching}$ of the switch, it is specified that

$$gnd = [0V, 2V),$$
$$batt = [8V, 10V),$$
$$pos_{switching} = 40\%.$$

The target distinctions are determined by the goal to distinguish between the ground voltage, corresponding to partition element $\{[0V, 2V)\}$, and battery voltage, corresponding to the partition element $\{[8V, 10V)\}$, for the domain of the variable v_{switch}:

$$\pi_{targ,v_{switch}} = \{\{[0V, 2V\}, \{[2V, 4V), [4V, 6V), [6V, 8V)\}, \{[8V, 10V)\}\}.$$

Based on these inputs, the component for computation of induced distinctions determines a partition for v_{pot} that consists of three partition elements (variable v_{pot} is denoted `Potentiometer.TV.vpot.voltage` in Raz'r's system description):

```
<Partition VARIABLE_NAME="Potentiometer.TV.vpot.voltage"
    <PartitionElement>
        <Value VALUE="[8V,10V)"/>
        <Value VALUE="[6V,8V)"/>
    </PartitionElement>
    <PartitionElement>
        <Value VALUE="[4V,6V)"/>
    </PartitionElement>
    <PartitionElement>
        <Value VALUE="[2V,4V)"/>
        <Value VALUE="[0V,2V)"/>
    </PartitionElement>
</Partition>
```

The first qualitative value $\{[0V, 2V), [2V, 4V)\}$ corresponds to situations where v_{switch} equals ground voltage, the third qualitative value $\{[6V, 8V), [8V, 10V)\}$ corresponds to situations where v_{switch} equals battery voltage, and the second qualitative value {[4V,6V)} corresponds to situations where the position of the switch and, hence, the voltage of v_{switch}, is ambiguous. If the three qualitative

values are denoted *gnd_switch*, *switching* and *batt_switch*, respectively, then the domain abstraction for v_{pot} is described as

$$v_{pot} = \begin{cases} gnd\_switch & : & voltage \in [0V, 4V) \\ switching & : & voltage \in [4V, 6V) \\ batt\_switch & : & voltage \in [6V, 10V) \end{cases}$$

AQUA induces also the partition

$$\{right\}, \{left\}$$

for the state of the switch and

$$\{60\%, 80\%, 100\%\}, \{0\%, 20\%, 40\%\}$$

for the domain of the pedal position. The resulting abstracted behavior model $SD_{transform}$ has a tuple space of 9216. The original model SD_{base}, in contrast, had a tuple space of approximately $5.6 \cdot 10^7$. Compared to a generic model $SD_{generic}$ that, as indicated in chapter 1, distinguishes only between the qualitative values *gnd*, *between* and *batt*, i.e.

$$v_{pot} = \begin{cases} gnd & : & voltage \in [0V, 2V) \\ between & : & voltage \in [2V, 8V) \\ batt & : & voltage \in [8V, 10V) \end{cases}$$

the derived model $SD_{transform}$ uses the same domain size for v_{pot} (i.e. the tuple space is equal), but it is more adequate in the context of the specified task. To see the differences between the three models SD_{base}, $SD_{transform}$ and $SD_{generic}$, consider the following example of an on-board diagnosis situation. Assume that the components of the pedal position sensor that can be faulty are the potentiometer, the switch, and the two nodes. Assume further that the control unit of the pedal position sensor receives a sequence of ten real-valued sensor readings as shown below:

t	v_{pot}	v_{switch}
0	$2.4V$	$0.5V$
1	$2.3V$	$0.5V$
2	$2.5V$	$0.5V$
3	$3.4V$	$0.4V$
4	$5.6V$	$0.4V$
5	$6.8V$	$0.5V$
6	$8.1V$	$9.7V$
7	$8.3V$	$9.6V$
8	$8.2V$	$9.8V$
9	$8.4V$	$9.7V$

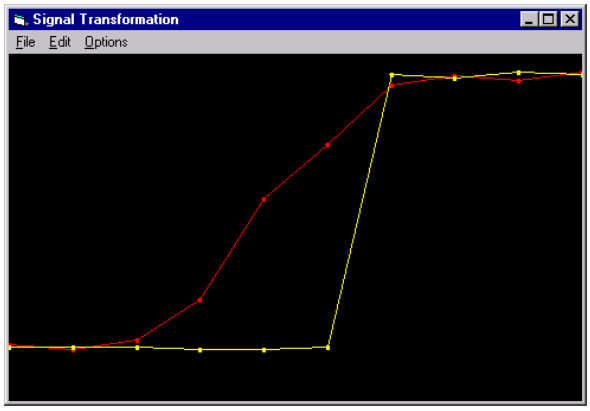

Figure 7.7: Example of Measurements for the Pedal Position Sensor (darker line shows signal of v_{pos}, brighter line shows signal of v_{switch})

The measurements reflect a delayed switch-over behavior of the switch component (see figure 7.7), as caused e.g. by a mechanical failure of this component. For the granularity of the base model SD_{base}, the signal transformation component yields four qualitative observations at time points $t = 0$, 4, 5 and 6, respectively:

t	v_{pot}	v_{switch}
0	$[2V, 4V)$	$[0V, 2V)$
4	$[4V, 6V)$	$[0V, 2V)$
5	$[6V, 8V)$	$[0V, 2V)$
6	$[8V, 10V)$	$[8V, 10V)$

For time point $t = 5$, the run-time system (RTS) detects an inconsistency of the observations with the model SD_{base}. The run-time for diagnosis is 0.455 seconds. It yields the conflict

```
{Potentiometer.ok, Switch.ok, Node1.ok, Node2.ok}
```

Under the domain mapping corresponding to the model $SD_{transform}$, i.e. the derived qualitative model, the signal transformation component also yields four qualitative observations at the same time points:

t	v_{pot}	v_{switch}
0	$[0V, 4V)$	$[0V, 2V)$
4	$[4V, 6V)$	$[0V, 2V)$
5	$[6V, 10V)$	$[0V, 2V)$
6	$[6V, 10V)$	$[8V, 10V)$

Model $SD_{transform}$ also detects the fault for time point $t = 5$, based on the inconsistency between $v_{pot} = batt\_switch$ and $v_{switch} = gnd\_switch$. The run-time for diagnosis is 0.256 seconds, i.e. about 43% less than for the fine-grained

model SD_{base}, which corresponds roughly to the reduction of the domain size of v_{pot}. $SD_{transform}$ also yields the conflict

`{Potentiometer.ok, Switch.ok, Node1.ok, Node2.ok}`

Finally, for the model $SD_{generic}$, the signal transformation component derives two qualitative observations at time points $t = 0$ and $t = 6$:

t	v_{pot}	v_{switch}
0	$[2V, 8V)$	$[0V, 2V)$
6	$[8V, 10V)$	$[8V, 10V)$

However, model $SD_{generic}$, though using the same number of qualitative values as SD_{base}, does not reveal the fault (i.e., it detects no conflicts), because the qualitative observation $v_{pot} = between$ is consistent with the observation $v_{switch} = gnd$.

This example illustrates AQUA's ability to support the modeling problem of determining qualitative values, i.e. distinctions in the domain of variables that are essential for a certain task. In this case, the task involved distinctions for the magnitudes of variables.

7.2.2 Determining Significant Deviations and Diagnostic Distinctions

Another interesting application of automated domain abstraction is to support the derivation of significant *deviations*. In some cases, what constitutes a significant distinction in a model of a component is not determined by the absolute values of other variables, but by the fact whether or not it enforces a significant deviation on them, regardless of what their specific value is. For instance, a view often taken in FMEA is that the function of the overall device imposes a certain "tolerance" on the output of this device, and its components are not considered faulty unless their behavior causes a disturbance of the output beyond the given tolerance. If we succeed to compute the tolerances of the parameters of the component models starting from the given functional specification, we can automatically generate qualitative models that reflect the particular device and its function. Section 4.3.2 has outlined such deviation models that capture deviations of the system behavior compared to some reference behavior.

The analysis of significant distinctions developed above can be applied to deviation models. We can specify what is considered to be a significant deviation of some relevant variables by target distinctions for the respective deviation variable. This will induce partitions for other deviation variables, but also for the magnitudes of the variables.

Induced distinctions can also be used to properly model the "border" that separates correct behavior from faulty behavior. The problem to be solved is the following: Given a set of behavior modes for each component (correct modes and faulty ones), what distinctions have to be made in the models to help discriminate among the modes? One way to address this problem is to represent

the models by including mode variables in the model relation, as described in section 3.6.3. The goal is then to determine the model granularity given target distinctions for the mode variables that separate the different modes. Basically, this means treating the behavior mode like a state variable whose values have to be completely distinguished from each other.

The following example illustrates both aspects: specifying target distinctions for modes, and deriving distinctions for variables that describe deviations.

Container Filling Example

Consider a simple container filling problem as illustrated in figure 7.8. The system comprises a reservoir (which is assumed to be never empty and not shown in the figure) filled with liquid with pressure p_{inlet}. It is connected via a valve with maximal diameter A_{max} to an outlet pipe with pressure p_{outlet} that fills a container with bottom area B and vertical walls. The level of liquid in the container is denoted l. The task is to use a model in order to design the control scheme that opens and closes the valve in order to fill the container up to a given height h with a precision of $\Delta h > 0$. It is assumed that the control scheme is binary, i.e. the valve can either be fully open or fully closed.

The example does not appear to be an application from the automotive domain at a first glance. However, consider it to be a simplification of a controlled injector (i.e., the valve) that is to supply a certain amount of diesel fuel to the combustion chamber (i.e., the container) of the car engine. Below h, the fuel mixture in the combustion chamber will be too lean in order to burn properly. Above $\Delta h + h$, there will be too much fuel to burn it completely, a situation that should be avoided in any case.

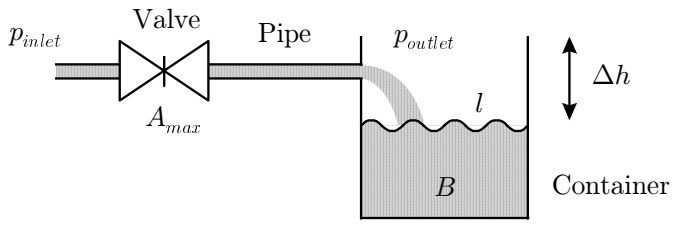

Figure 7.8: Filling a container

All the components in the system are fairly standard, and we can easily specify their behavior models in order to compose a base model of the example system. The container component has two states, one that captures the situation where the container level is within the specified range and one that represents overflow:

$$overflow : l > h + \Delta h,$$
$$\neg overflow : l \leq h + \Delta h.$$

The valve model associates the equations

$$closed : q_{valve} = 0$$
$$open : q_{valve} = A_{max} \cdot sgn(p_{inlet} - p_{outlet})\sqrt{|p_{inlet} - p_{outlet}|}$$

with the states open ($A = A_{max}$) and closed ($A = 0$) of the valve, where q_{valve} denotes the flow through the valve. But what about the transitions between the states? For the valve, the transition between open and closed is not instantaneous, but it requires a certain amount of time (usually referred to as the "dead time" of the valve), during which some amount of liquid v_{dead} is pouring into the container:

$$closing : v_{dead} \leq \Delta t \cdot A_{max} \cdot sgn(p_{inlet} - p_{outlet})\sqrt{|p_{inlet} - p_{outlet}|}.$$

The question is whether or not we have to consider this delay in our model. I.e., is it necessary to model the dead time of the valve, or can the behavior model of the valve be formulated using instantaneous transitions between the states open and closed, such that the duration of the closing operation is neglected?

Usually, such a modeling question will be decided ad hoc by a human modeler, e.g. relying on the fact that the valve will be closing "reasonably fast". From a general and systematic point of view, however, the answer depends on the targeted precision Δh and on the characteristics of the entire configuration, namely p_{inlet}, p_{outlet}, A_{max}, and B. For instance, perhaps the container is so large that the increase of the height during closing of the valve is negligible. Perhaps, the time required for closing the valve is quite significant, because the required precision Δh is tight or because the pressure difference $p_{inlet} - p_{outlet}$ is high.

Given these contextual conditions of the task, a human modeler of the system would be able to manually determine a threshold for durations of the opening that determines the boundary between significance and insignificance with respect to the required precision Δh. Based on the result, he could then decide whether or not it is appropriate to approximate a continuous valve model by one with discontinuous transitions. Note that the result is also influenced by the precision of the inputs to the calculation. For instance, if it is just known that the diameter is between zero and A_{max} during opening of the valve, a boundary on the duration is obtained which is smaller than the one calculated for an opening that is described as linear within a certain tolerance. In the same way, the precision of the pressure values influences the distinctions on the duration.

In the following, we present how our method can automatically derive such distinctions for an instance of the container filling problem. The domain for all pressure, flow and parameter variables in the base model is, for the sake of simplicity, assumed to be the same 19-valued domain depicted below, which consists of open intervals and points between them:

$$DOM = \{(-\infty, -4), -4, (-4, -3), -3, \ldots, (3, 4), 4, (4, \infty)\}.$$

We further assume that all variables are observable at the granularity of this base domain. The model SD_{base} formulated in terms of this domain consists of

12 variables and 2 mode variables, which means it has a tuple space (i.e. size of $DOM(\boldsymbol{v})$) of $19^{12} \cdot 2^2 \approx 8.9 \cdot 10^{15}$. The target distinction for the example can be stated by giving a distinction for the state variable of the container component:

$$\pi_{targ,Tank.mode} = \{\{overflow\}, \{\neg overflow\}\}.$$

What we are then after is an answer to the question whether or not we have to include the transition delay Δt of the valve's closing operation in the model or not. The following result is obtained for parameters A_{max}, Δh, B, p_{inlet} and p_{outlet} given as

$$\begin{gathered} A_{max} = (0,1), \\ \Delta h = (2,3), \\ B = 1, \\ p_{inlet} \in \{(2,3), 3, (3,4)\}, \\ p_{outlet} = 0. \end{gathered}$$

After computation of the model relation for SD_{base}, 21036 tuples remain consistent for the valve component. AQUA determines the following induced distinction for the valve delay parameter Δt (denoted `Valve1.PV.Delay` in the system description):

```
<Partition VARIABLE_NAME="Valve1.PV.Delay">
    <PartitionElement>
        <Value VALUE="(1,2)"/>
        <Value VALUE="(4,inf)"/>
        <Value VALUE="4"/>
        <Value VALUE="(3,4)"/>
        <Value VALUE="3"/>
        <Value VALUE="(2,3)"/>
        <Value VALUE="2"/>
    </PartitionElement>
    <PartitionElement>
        <Value VALUE="(-inf,-4)"/>
        <Value VALUE="1"/>
        <Value VALUE="(0,1)"/>
        <Value VALUE="0"/>
        <Value VALUE="(-1,0)"/>
        <Value VALUE="-1"/>
        <Value VALUE="(-2,-1)"/>
        <Value VALUE="-2"/>
        <Value VALUE="(-3,-2)"/>
        <Value VALUE="-3"/>
        <Value VALUE="(-4,-3)"/>
        <Value VALUE="-4"/>
    </PartitionElement>
</Partition>
```

The first partition element corresponds to situations where overflow of the container is possible (though will not necessarily occur). The second partition element corresponds to situations where overflow is not possible. This means that all situations where the delay during closing the valve is part of the second partition element, i.e. $\Delta t \leq 1$, are equivalent to zero delay, i.e. $\Delta t = 0$. In other words, the dead time of the valve can be neglected in the behavior model of the valve if it belongs to the second qualitative value, and must be considered in the model if it belongs to the first qualitative value.

To give an idea of the problem size, a multiplication constraint that uses the base domain consists of 713 tuples out of a tuple space of $19^3 = 6859$, which amounts to a ratio of 10.4 % consistent combinations of values. For a domain that consists just of signs, the respective constraint would have 9 tuples out of a tuple space of $3^3 = 27$, i.e. a ratio of 33.3 % consistent tuples. The constraint describing the valve equation, for instance, has 23075 tuples. Compared to a tuple space of $8.9 \cdot 10^{15}$ for the base model SD_{base}, the size of the tuple space $\tau_{ind}(DOM(\boldsymbol{v})$ of the abstracted behavior model $SD_{transform}$ is only $1.2 \cdot 10^4$.

As a modification of the task, we can also start with a given parameter for the delay and ask for a partition for the mode variable of the valve component, given the same target distinction for the container. Depending on the magnitude of the delay, it will turn out to be necessary or unnecessary to explicitly distinguish the behavior modes “closing” and “closed” of the valve. For the given parameters, we obtain the partition:

```
<Partition VARIABLE_NAME="Valve1 mode">
    <PartitionElement>
        <Value VALUE="closing"/>
        <Value VALUE="closed"/>
    </PartitionElement>
    <PartitionElement>
        <Value VALUE="open"/>
    </PartitionElement>
</Partition>
```

I.e. the distinction between the behavior modes becomes irrelevant, if and only if the parameter for delay is restricted to the first qualitative value.

This example illustrates AQUA’s generality in the sense that it is not limited to deriving distinctions for magnitudes of variables, but can also be applied to modeling problems involving behavior modes and deviations of variables.

7.2.3 Supporting Diagnosability Analysis and Design

Another important task during the design of a physical system is diagnosability analysis. Diagnosability analysis means to decide, for a given design of a physical system, whether the available observations are sufficient in order to discriminate between possible behaviors of the system, which is relevant e.g. for FMEA (see chapter 2).

AQUA can also be useful for this task. Like in section 7.2.2, the basic principle is that distinctions between possible behaviors can be treated as target distinctions for the behavior modes or states of components. The problem can then be cast as the question whether the granularity of the available observations is sufficient for the target distinctions we want to make.

Consider the following modification of the container example presented in section 7.2.2. Assume we know that the transition delay Δt of the valve lies between zero and two:

$$0 < \Delta t < 2.$$

Because we consider the distinction between overflow and non-overflow of the container component (which, in the context of automotive systems, can be interpreted as complete vs. incomplete fuel combustion) to be of great importance, we want to augment the system by a sensor that measures the pressure p_{inlet} of the liquid in the reservoir. Based on this sensor signal, we then want to decide whether overflow can occur or not. Assume that we have the option to use a cheap but inaccurate sensor that can only determine if the pressure is below the threshold value 3 or not:

$$\pi_{obs,p_{inlet}} = \{\{(-\infty, -4), \ldots, (2, 3)\}, \{3, \ldots, (4, \infty)\}\}.$$

The question is then whether this observable granularity is actually helpful in order to decide whether the tank is overflowing or not. Using the terminology of chapter 5, this question can be interpreted as the task of determining whether the target distinction between the two states of the container is observable or not.

We can use AQUA in order to answer this question. To this end, we have to use the same target partition as specified in section 7.2.2, and use the partition above as the only observable distinction. AQUA determines that the granularity of the sensor as shown above is not sufficient in order to observe either an overflowing or a non-overflowing container:

```
For variable 'Tank1 mode', the target partition
is not observable.
```

This means that for both of the possible sensor readings, either of the two behaviors is consistent. Based on this result, we might decide to employ a more accurate sensor instead. For instance, if we have a granularity of observations for p_{inlet} that distinguishes all positive elements of the base domain from each other, it turns out that the target distinction becomes observable. This example, though simple, illustrates that AQUA can also be helpful in supporting the evaluation of possible design alternatives of a physical system.

7.2.4 Deriving Qualitative Abstractions of Real-valued Models

In the examples that we have considered up to now, the domains of the variables in the base model were finite. However, this might not always be the case, especially

in the context of industrial applications. Instead, as outlined in chapter 2, real-valued models or so-called hybrid models that incorporate a continuous and a discrete part are quite common in the automotive industry, e.g. in order to perform behavior analysis through numerical simulation. Therefore, the question of how to exploit real-valued behavior models or behavior model fragments in the context of problem solving is of great practical importance.

In this section, we describe how AQUA can be used in order to make use of real-valued behavioral knowledge for automated qualitative modeling. It was already noted in section 7.1.5 that AQUA includes a software component for the generation of system descriptions that can derive a model with finite domains starting from a system description that includes real-valued constraint types taken from a generic set of algebraic operators. Real-valued behavioral information that is more specific and difficult to express in the form of algebraic constraints, such as e.g. a characteristic map, can be provided in the form of a real-valued parameter description (chapter 8 presents an example of a parameter description corresponding to a characteristic map).

The basic approach to make base models including infinite domains amenable to model-based reasoning is to automatically transform them, using AQUA, to a finite model through an initial domain abstraction from the real numbers to some finite domain with appropriate grain size. Model-based problem solving such as behavior prediction or diagnosis can then be performed using this abstracted model. This is illustrated in figure 7.9.

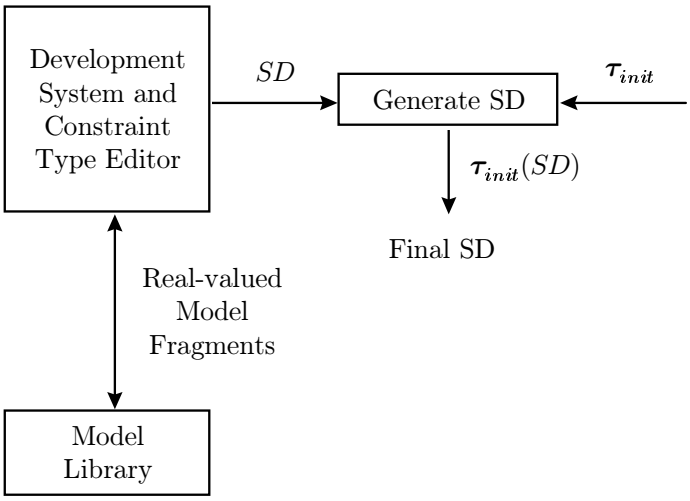

Figure 7.9: Abstracting real-valued base models using AQUA

However, this leaves one with the question of how to choose a domain mapping with appropriate grain size that does the initial transformation from the real numbers to finite domains. If the initial abstraction is too coarse, we might lose some behavioral features of the original model that are essential for the task we

want to solve, and the transformed model will be of limited use. On the other hand, if the initial abstraction is too fine-grained, we obtain a system description that contains unnecessarily large domains and thus unnecessarily large constraint types, potentially rendering the further steps of model-based problem solving with this model cumbersome or even infeasible.

AQUA can be helpful in this context. For the latter case, i.e. a too fine-grained initial model, it is obvious that we can use task-dependent qualitative model transformation in order to get rid of any "useless" distinctions in the model, just as we did for the examples in the previous sections. However, for the first case, i.e. a too coarse initial model, we cannot solve the problem by further abstracting the model, because the relevant information has already been abstracted away, i.e. it is "lost" in the initial model. Hence, in this situation, the initial domain mapping has to be further *refined* instead of further abstracted.

But also in this case, AQUA can be helpful. The algorithm presented in chapter 6 derives information about the solutions at the level of target distinctions that a certain domain element is consistent with. This information can be exploited in order to decide which parts of the initial domain mapping should be refined. For instance, if a domain element is consistent with a large number of solutions, further refinement of this domain value can help to increase the possibilities of discriminating among the solutions. This idea leads to a scheme for qualitative abstraction of real-valued models that is depicted in figure 7.10.

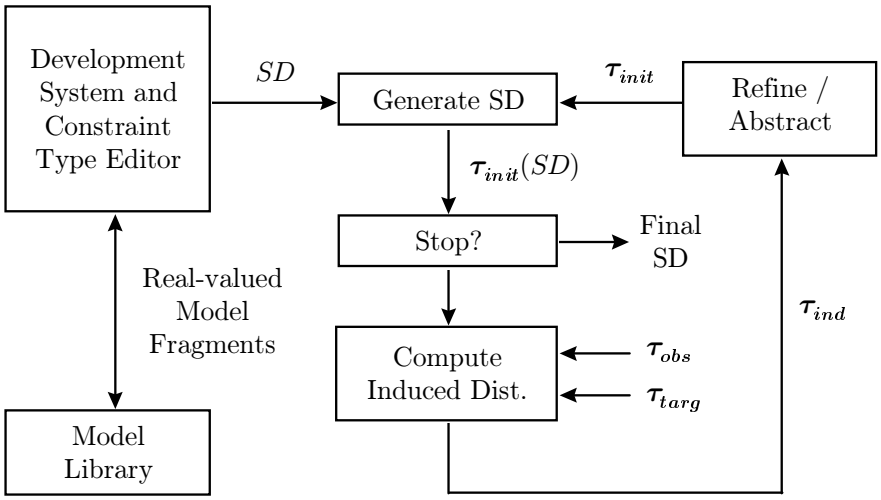

Figure 7.10: Iterative domain refinement using AQUA

In the following, this approach is referred to as *iterative domain refinement* of models with real-valued base domains. For the iteration steps, different strategies are possible in order to refine the initial domain abstraction. In the following, we present an instance of this strategy that is based on repeatedly bisecting domain values that are consistent with a maximum number of solutions.

Iterative Domain Refinement for the Pedal Position Sensor Example

We present an application of iterative domain refinement for the pedal position sensor device that was presented in section 1.1.1. The base model SD_{real} for the pedal position sensor consists of the model fragments as shown in section 3.4. However, the domains of the variables are now interpreted as being equal to the real numbers. As initial domain abstraction, we start with the same domains as specified in section 7.2.1, i.e. the initial domain for v_{pot} is

$$DOM_0 = \{[0V, 2V), [2V, 4V), [4V, 6V), [6V, 8V), [8V, 10V)\}.$$

The resulting initial model, termed SD_0, is equal to SD_{base} defined in section 7.2.1. For simplicity of the presentation, in the following we concentrate on refining the domain of v_{pot} only, and we assume that all variables are observable at the granularity of their base domain. Based on the target distinction for v_{switch}, AQUA derives the following partition for v_{pot} (see figure 7.11a):

$$\pi_{ind,v_{pot}} = \{\{[0V, 2V), [2V, 4V)\}, \{[4V, 6V)\}, \{[6V, 8V), [8V, 10V)\}\}.$$

The domain elements $\{[0V, 2V), [2V, 4V)\}$ and $\{[6V, 8V), [8V, 10V)\}$ are consistent with one solution each, whereas the domain element $\{[4V, 6V)\}$ is consistent with two solutions. Therefore, we decide to split the domain element $[4V, 6V)$ of DOM_0 into two new domain elements $[4V, 5V)$ and $[5V, 6V)$, whereas we aggregate the other domain elements of DOM_0 according to the partition. After this first step, the domain of v_{pot} becomes

$$DOM_1 = \{[0V, 4V), [4V, 5V), [5V, 6V), [6V, 10V)\}.$$

This modified domain abstraction leads to a modified model SD_1 of the pedal position sensor. Using SD_1 and the target distinction for v_{switch}, AQUA derives the following induced domain for v_{pot} (see figure 7.11b):

$$\pi_{ind,v_{pot}} = \{\{[0V, 4V)\}, \{[4V, 5V), [5V, 6V)\}, \{[6V, 10V)\}\}.$$

We observe that the partition element $\{[4V, 5V), [5V, 6V)\}$ is consistent with two solutions, hence each of the new domain values is consistent with a maximum set of solutions. Therefore, we have to further split one of the two domain values that where newly introduced in DOM_1. As an arbitrary choice, we bisect $[4V, 5V)$ to obtain two new domain elements $[4V, 4.5V)$ and $[4.5V, 5V)$. After this second step, the domain of v_{pot} becomes

$$DOM_2 = \{[0V, 4V), [4V, 4.5V), [4.5V, 5V), [5V, 6V), [6V, 10V)\}.$$

Running AQUA with the corresponding model SD_2 yields (see figure 7.11c)

$$\pi_{ind,v_{pot}} = \{\{[0V, 4V), [4V, 4.5V)\}, \{[4.5V, 5V), [5V, 6V)\}, \{[6V, 10V)\}\}.$$

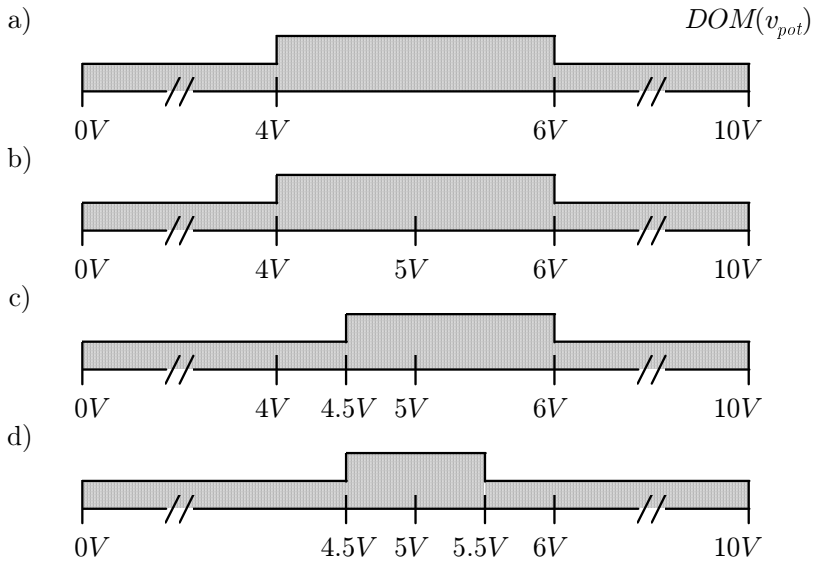

Figure 7.11: Iterative domain refinement for the domain of v_{pot} in the pedal position sensor example (boxes indicate the partition elements derived by AQUA, the height of a box indicates the number of solutions for the partition element)

If we further split the largest domain value $[5V, 6V)$ that is consistent with a maximum number of solutions into $[5V, 5.5V)$ and $[5.5V, 6V)$, we get the following modified domain for v_{pot}:

$$DOM_3 = \{[0V, 4.5V), [4.5V, 5V), [5V, 5.5V), [5.5V, 6V), [6V, 10V)\}.$$

Running AQUA with the corresponding model SD_3 yields (see figure 7.11d)

$$\pi_{ind,v_{pot}} = \{\{[0V, 4.5V)\}, \{[4.5V, 5V), [5V, 5.5V)\}, \{[5.5V, 6V), [6V, 10V)\}\}.$$

Let $SD_{iterate}$ denote the behavior model corresponding to this partition. In $SD_{iterate}$, the domain for v_{pot} consists of the following three values:

$$v_{pot} = \begin{cases} gnd\_switch' & : & voltage \in [0V, 4.5V) \\ switching' & : & voltage \in [4.5V, 5.5V) \\ batt\_switch' & : & voltage \in [5.5V, 10V) \end{cases}$$

Note that the size of the model in terms of the tuple space has not increased during the iterative refinement, as for each of the steps, the domain of v_{pot} did not contain more than the original five elements. Thus, each iteration takes roughly the same computation time for AQUA, independent of the current iteration stage. Only the interpretation of the domain values, i.e. the mapping from real numbers to the domain values, is changing. The model gets thus more and more adapted to the task it is intended for, as specified by the target distinction for v_{switch}. This

can be evaluated using the diagnosis example outlined in section 7.2.1. For the granularity of the model $SD_{iterate}$, the signal transformation component yields three qualitative observations at time points $t = 0$, 4 and 6:

t	v_{pot}	v_{switch}
0	$[0V, 4.5V)$	$[0V, 2V)$
4	$[5.5V, 10V)$	$[0V, 2V)$
6	$[5.5V, 10V)$	$[8V, 10V)$

For time point $t = 4$, the run time system detects a discrepancy between the observations for v_{pot} and v_{switch} and the model $SD_{iterate}$. The resulting conflict is the same that has been derived for SD_{base} for time point $t = 5$. Hence, based on the reduced ambiguity for the domain of v_{pot}, $SD_{iterate}$ detects the same fault for the pedal position sensor, but one time point earlier than the model SD_{base} and any abstractions thereof.

This example illustrates how AQUA offers a principled way to exploit real-valued models, as e.g. developed by tools such as Matlab/Simulink or MatrixX, in order to make them amenable to model-based reasoning methods. In this way, AQUA can be seen as a contribution to bridging the gap between quantitative and qualitative modeling, a problem that has been identified in chapter 2 as one of the major roadblocks to a more wide-spread use of model-based reasoning techniques.

This section is also meant to illustrate the flexibility of AQUA in the sense that further extensions like iterative domain refinement can be designed and implemented "on top" of the AQUA components. Currently, the iteration steps (i.e., refining the domains based on the results of AQUA) have been performed manually. An interesting direction for further work would be to automate this process (see also chapter 9).

7.3 Discussion

The examples in this chapter have shown how AQUA can provide automated support for a number of difficult tasks related to modeling and building model-based systems, which previously had to be carried out manually or solved on an ad hoc basis. The common and concise theoretical basis is to find suitable domains for the variables in a model. However, in different contexts this basic task can have different interpretations, depending on what the terms variable and domain refer to, including magnitudes, modes of components, and deviations from reference behaviors. The example in section 7.2.2 required to find an appropriate granularity of time, which is related to the problem of time-scale abstraction ([Iwa92, Kui94], see also section 4.5.4). AQUA provides a general framework for these different applications.

AQUA meets also important practical requirements. The existing Raz'r system allows to define finite domains and constraint types to be used within a behavior model. However, committing early to specific domains is problematic

since it is difficult to choose the right ones ad hoc, and there is no way of adapting them except for modifying their definition. An effect of automated domain abstraction in this context is that it "hides" — at least to a certain extent — the definition of domains from the user, thereby increasing the transparency of the behavior models. AQUA can thus be viewed as an extension to Raz'r that allows to pursue a more natural and more flexible way of modeling that removes from the user some of the burden of thinking in terms of the constraint representation and instead allows him to concentrate more on the task and the available knowledge. Ideally, as demonstrated in the last section, the ground domains do not have to be fixed et all, i.e. they can be equal to the real numbers, which constitutes the standard way how engineers formulate their models.

The software components that realize this functionality have been presented in sections 7.1.5 through 7.1.7. Although equipped with full-featured GUIs and making use of the Raz'r interfaces, they are still prototypic implementations. For the generation of system descriptions (section 7.1.5), possible improvements include the identification of constraint types within the generated model. Transformed system descriptions currently consist only of instances of constraints instead of constraint types. However, experiments indicate that for larger models, the identification of types of constraints could be worthwhile. The component for the computation of induced distinctions (section 7.1.6) rests on Raz'r's constraint compiler. The performance of this component could immediately benefit from improvements of the constraint compiler; currently, it uses only a non-optimized implementation of OBDDs to represent and manipulate constraint types. Another important direction for extending this component concerns the integration into the model revision process. Currently, all behavior modes of a component that are defined in the system description will be included in the SD Tree. In terms of the spectrum defined in section 3.6.3, all possible model revisions are anticipated, and consequently, the resulting model relation can be quite large. More flexibility (and, perhaps, increased performance) could be achieved by initially including only some of the behavior modes, and performing the computation of induced distinctions for the remaining modes only in the case where the corresponding revisions are really necessary. Finally, the signal transformation component (section 7.1.7) could be extended by including further import filters for additional data file formats or augmenting the possibilities for signal preprocessing.

For the examples, the definitions of domain values frequently involved intervals over the real numbers. This might lead to the impression that the obtained results are essentially an application of — or could be reproduced by — interval arithmetic. However, AQUA uses intervals only to specify domain mappings that abstract the set of real numbers. Intervals are an intuitive and powerful means to define subsets of the real numbers, and thus they are employed for this purpose. Once the abstraction is done, however, further reasoning is based on the abstracted model, which is independent of the definition of the domain mapping. Therefore, well-known problems of interval-based arithmetic, such as cumulation of interval "splitting" during propagation (see [RR84], [Str90]), do not occur in this approach, as it never combines intervals with each other.

7.4 Summary

This chapter described the implementation of task-dependent qualitative abstraction in the form of three software modules. It showed how they are integrated into a general (commercial) framework for model-based reasoning and diagnosis. A number of principled examples illustrated how the capabilities of this enhanced framework can be used to support automated modeling and building of model-based systems. The principled examples leave open the question whether the solution scales up to the real-world applications that where described in chapter 2. Therefore, the next chapter will describe a real-world application from the domain of on-board diagnosis for automotive systems.

Chapter 8

Real-world Application: On-board Diagnosis of a Passenger Vehicle

The purpose of this chapter is to demonstrate the application and integration of the software framework AQUA in the context of a real-word application. The application we consider in this chapter is the development of a prototype for model-based on-board diagnosis of an electronic diesel control system of a passenger vehicle. This system is part of the application domain that was outlined in chapter 2.

The resulting prototype constitutes the first model-based diagnosis system that runs on-board a passenger vehicle. It demonstrates the ability of the approach to support the generation of appropriate models for a typical automotive system. Evaluation using measurements from a demonstrator car shows that the prototype performs favorable compared to traditional on-board diagnostic approaches, as it allows to diagnose failures that are not diagnoseable or, at least, hard to diagnose with the existing on-board diagnosis functionality of the ECU.

8.1 Background and Motivation

As outlined in chapter 2, increased environmental awareness poses stricter constraints on car manufacturers to develop clean cars and also to keep them clean during their life cycle. These growing constraints are reflected in increased requirements for on-board diagnosis development for passenger vehicles.

In response to this situation, several car manufacturers and suppliers joined to launch the Brite-EuRam project VMBD (Vehicle Model Based Diagnosis) with the intention to promote the transfer of model-based reasoning technology by the challenge of applying it to diagnosis of series passenger cars.

Subgroups within the VMBD project have focused on different problems in connection with vehicle diagnosis, such as off-line compilation of a model to use it in an on-board environment and experimenting with additional sensors in the

diesel injection system ([CCG+99]), or using a numerical approach for off-board diagnosis of faults in the hydraulic and mechanical parts of an automatic transmission system ([BTC+99]).

One subgroup involving Volvo Car Corporation, Robert Bosch GmbH and OCC'M Software GmbH was formed in order to develop a prototype of a model-based system that is capable of diagnosing problems related to increased carbon emissions of diesel engines, a problem of significant importance with respect to environmental impact and compliance with legal requirements. This system had to make use of the sensor signals that are available on-board, transform them to a qualitative level and exploit them for detecting and localizing faults based on a model of the turbo control system of the diesel engine. The system was installed on a Volvo demonstrator vehicle with a number of built-in faults.

8.1.1 Demonstrator Car

For the subgroup of the above-mentioned project, a Volvo 850 TDI demonstrator car was made available for hands-on experimentation with the DTI application. Failures were induced in the car during various operational conditions of the engine with a measurement acquisition system running, which also allowed to inspect the results of the conventional diagnostic capabilities of the control unit. The various failures in the demonstrator car could be adjusted by potentiometers and triggered by switchboards from inside the passenger compartment (see figures 8.1 and 8.2). A pneumatic leakage in a pipe, for example, was implemented by additional valves that could be opened and closed by electrical switches.

For these experiments, additional interfaces and devices had to be installed in the vehicle. At present, control units still have rather limited computing power which prevents the integration of the model-based diagnosis system within the ECU software. To circumvent this restriction, a so-called application control unit was used in the demonstrator. Application control units are normally used for calibration of ECU software for a specific vehicle type. They are equipped with special dual-ported memory chips such that, in principle, all variables and signals of the control unit are accessible in real-time, without interfering the normal operation of the ECU. The data acquired from the vehicle was interfaced to the model-based diagnosis prototype, which was running on a portable PC inside the passenger compartment (see figure 8.3). In figure 8.3, ETK is a hardware interface closely attached to the application ECU providing access to its controller bus. MAC is a protocol conversion box which stores the information gathered from the ETK, while VS100 (see also section 7.1.4) is a commercial tool that car suppliers use for acquisition, storage, and display of control unit data. It runs on the same portable PC as the on-board diagnostic prototype. The AD-Scan device and the PC Tester allow to read in further signals (dotted lines) from additional sensors or workshop equipment for the purpose of off-board diagnosis in the VMBD project.

Although this means that the model-based diagnostic software is not really running embedded within the ECU, this solution was considered adequate for the case studies since it provided all important constraints except the space and

Figure 8.1: View of the Volvo Demonstrator Car showing the notebook connected to the ECU

computing power limitations of the ECU. As noted in chapter 2, this latter aspect is beginning to be more and more relaxed in practice, anyway.

8.1.2 Application System

The demonstrator vehicle used in the VMBD project was equipped with a so-called distributor-type injection (DTI) system ([Bau96]). The DTI is an approved system which has been on the market for many years. However, increased legislative and customer demands have lead to new requirements especially for aspects related to emissions and performance of this system. Figure 8.4 shows part of the system which is responsible for supplying air to the diesel engine. It can be decomposed into the exhaust gas re-circulation (EGR) subsystem (upper part of figure 8.4) and the turbo control subsystem (lower part of figure 8.4).

The purpose of the exhaust gas re-circulation system is to return a certain amount of the exhaust gas to the intake air to decrease the oxygen rate of the intake air and thus to reduce emission levels of the fuel combustion. Depending on driving conditions, the ECU governs the EGR converter to achieve a certain air pressure in a control pipe, which in turn sets the position of the exhaust gas re-circulation valve. The position of this re-circulation valve then determines how much of the exhaust gas is fed back to the air intake pipe.

The turbo control subsystem consists of a turbocharger turbine, which is driven by the engine's exhaust gas, for compressing — and thereby increasing the mass of — the air taken into the engine. The ECU controls the boost pressure

Figure 8.2: The glove compartment of the Volvo Demonstrator Car which contained the switchboard for controlling the built-in faults

(i.e., the pressure in the engine intake pipe) admitted in a certain driving situation by opening or closing the turbo control valve, which determines the position of a so-called wastegate valve. The position of this valve determines how much of the exhaust gas drives the exhaust turbine of the turbocharger.

8.1.3 Diagnostic Scenarios

The particular interest of the involved car manufacturer concerned failures of the DTI system that could not be captured or were found hard to capture by traditional on-board diagnosis. A major class of such problems is characterized by effects related to emissions and reduced engine performance due to an excessive quantity of fuel injected or insufficient airflow to the engine. Such failures lead to incomplete fuel combustion and increased carbon emissions due to non-burnt particles, and are, therefore, often called "black smoke" problems.

For example, one scenario in the demonstrator car consisted of a leakage in the air hose between the turbine outlet and the engine intake manifold. The scenario was realized in the car by installing an electric motor opening a valve to release pressure from the inter-cooler system via a 1.2 cm opening. If the leakage is opened, air mass is lost after having passed the air mass sensor. The fuel quantity calculated by the control unit based on this signal will therefore be too high for the actual amount of oxygen in the combustion chamber. This leads to incomplete combustion of the diesel fuel, which causes increased carbon emissions in the exhaust gas and reduces the torque of the engine. This effect is

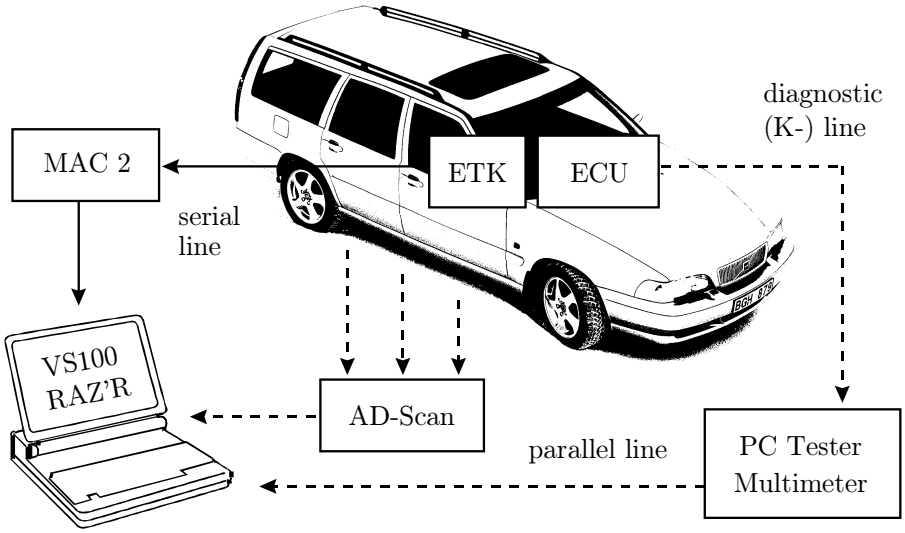

Figure 8.3: Architecture for data acquisition in the demonstrator car

perceivable for the driver as black smoke emerging from the exhaust system.

In another scenario, a wrong flow from the exhaust gas re-circulation system occured due to a faulty signal or mechanical failure in the EGR valve. The real fault installed in the car consisted of a switch used to control a magnetic valve that allows ingress of atmospheric pressure in the EGR valve, thus causing it to open outside its normal operating region. The rest of the installed scenarios involved faults in the boost pressure sensor, airflow sensor and engine temperature sensor. These faults were injected in the car by electrically manipulating the respective signal to the control unit. Table 8.1 lists the failure scenarios that have been installed in the demonstrator car.

8.1.4 Goals and Requirements

Black smoke presents a serious environmental problem for the DTI system, but can also impair safety due to possible thermic overload of the engine. Currently, the control unit of the DTI is equipped only with a restricted form of on-board diagnostic capabilities in order to detect this class of failures. It continuously monitors part of the sensor signals using predefined range and plausibility checks (see section 2.2.2) and is able to detect a limited number of faults on this basis. Quite often, however, it fails to discriminate among the different possible causes that lead to a failure. In many cases, it cannot even detect it at first hand.

One way how engineers concerned with this system try to improve this situation is to exploit more of the interdependencies between the signals. But currently, this is not done in a general and systematic way, leading to non-optimal solutions. The major reason for this is that faults related to black smoke are hard

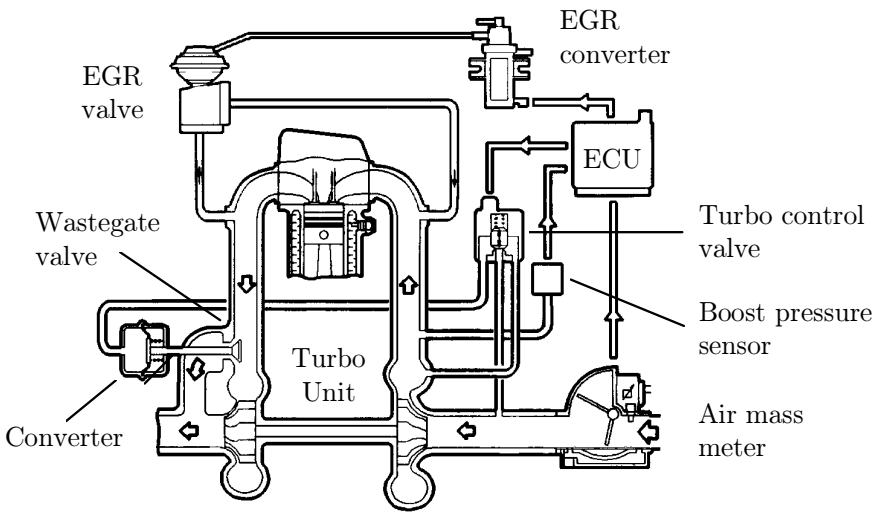

Figure 8.4: Turbo Control and exhaust gas re-circulation subsystem of the DTI

to capture using local signal range checks only, since the fault typically involves a wrong ratio of the inputs to the engine as reflected by the airflow signal, the boost pressure signal, and the amount of fuel injected. This means that the magnitudes of these signals are consistent for themselves, and detecting and localizing faults requires to take into consideration the behavior of several different signals, and therefore components of the system, at once. This motivates a model-based approach in order to exploit the analytic redundancy of the model in a systematical and automated way. Hence, in accordance with the overall thrust of the project, the goal was to

- develop a library of model fragments of the relevant components,
- generate, in a systematic way, a model of the DTI that is adequate for diagnosing faults in the diesel engine based on the sensor signals that are available to the ordinary ECU,
- perform model-based diagnosis with the resulting model, using real measurements taken from the demonstrator vehicle,
- compare the results with the ordinary ECU diagnostic capabilities and assess the real-time capabilities of this solution.

In the following sections, we describe how the behavior of the DTI system can be described by composing model fragments and how a qualitative model of the DTI can be obtained by applying qualitative model abstraction to this base behavior model. The on-board diagnosis prototype itself then consists of the module for transforming the raw sensor signals into qualitative observations (as

Scenario	Description	Physical realization of failure in the car
1	Leakage in air intake manifold	Additional valve installed in the manifold that can be opened and closed through an electric motor
2	Boost pressure sensor signal too high	Electrical manipulation of the pressure signal by increasing the resistance of the sensor through additional potentiometers
3	Airflow sensor signal too high	Electrical manipulation of the airflow signal by increasing the resistance of the sensor through additional potentiometers
4	EGR valve opening outside normal operating region	Additional magnetic valve installed in the EGR system that can be opened and closed through an electrical switch
5	Engine temperature signal too low	Electrical manipulation of the temperature signal by increasing the resistance of the sensor through additional potentiometers

Table 8.1: Failure scenarios installed in the demonstrator car

presented in section 7.1.7) and the model-based run-time system (RTS) of Raz'r that performs diagnosis for the derived model on the basis of these observations (see section 7.1.1).

8.2 Model Fragments

In this section, we present model fragments that describe the behavior of components occurring in the DTI air intake system. Since diesel engines are of wide-spread use, there exists a considerable amount of work on modeling the behavior of such components, e.g. [Hey88, Kea93, Sto97]. The model fragments in this section more specifically build on numerical behavior models described in [NN97, NN98] that originally have been developed for air intake components of a gasoline engine. However, following the requirements outlined in chapters 3 and 4, care has to be taken in order to ensure that the behavior descriptions are local to the component types, and that coverage of their physical behavior is achieved.

The presented behavior descriptions are based on a number of simplifying assumptions. The first assumption is that an iso-thermic model of the air intake system is sufficient. This means that temperature is not considered to be varying in the model. For the air intake part of the system, this is justified by the fact that the turbo charger component actually consists of a turbo charger turbine and an intercooler which keeps the temperature of the intake air nearly constant. The second assumption is that it is sufficient to capture the mean-cycle behavior of the system. This means that in-cycle variations of the combustion engine will not be considered in the model. The justification for this is that in-cycle variations and the effects of the considered faults appear on very different time

scales. The time scale of in-cycle variations is in the order of milliseconds, while the time scale of perceivable effects of the considered faults is in the order of seconds or minutes. The third and perhaps most severe simplification we make is that the effect of exhaust gas recirculation, i.e. variations in the oxygen rate of the intake air, will not be considered in the model. This latter restriction is due to the employed component-oriented ontology (see also section 3.3).

Since the DTI system is responsible for delivering air to the diesel engine, the behavior of components in terms of pneumatics is of central interest. An important physical law in this context is the ideal gas law ([Kea93, Bau96]), which captures the relationship between pressure p, volume V, mass m, and temperature T:

$$p \cdot V = m \cdot R \cdot T$$

The constant R is a substance-specific constant which can be derived from the universal gas constant. For air,

$$R = 0.287 \frac{J}{g \cdot K}$$

Due to the assumption about iso-thermicity, the factor T is kept constant. T is specified as 298 K.

8.2.1 Engine Model

The engine model is the most complex part of the DTI behavior model. It consist of three different parts that capture different behavioral aspects of the diesel engine: its mechanical behavior, its combustion behavior, and its performance behavior. The terminal variables of the engine component are given by the engine's crankshaft speed n_{engine} [1/min], inlet and outlet pressure p_{inlet}, p_{outlet} [kPa], inlet and outlet air mass flow q_{inlet}, q_{outlet} [g/s] and injected fuel mass m_{fuel} [g/s].

Mechanical Behavior

This constraint type of the engine model describes the mechanical behavior of the engine in terms of the relationship between engine speed n_{engine}, the pressure at the inlet p_{inlet}, and the inlet air flow q_{inlet}. The latter variable q_{inlet} corresponds to the amount of air that is available for fuel combustion.

Following [Nyb99], q_{inlet} can be described as a function

$$q_{inlet} = f_{mech}(n_{engine}, p_{inlet})$$

of engine speed and pressure at the inlet that is monotonic at least for the standard driving situations. However, the function f_{mech} is highly non-linear. Because it depends on factors such as the turbulence of the air flow and the geometry of the engine, it is very hard to describe it in terms of mathematical equations. In fact, there is no known analytical model for f_{mech}.

Thus, we take an approach similar to the one in [Nyb99]. The function f_{mech} is captured as a *characteristic map* that is obtained from a number of measurements taken in different driving situations. There is a difficulty involved in these measurements as without additional sensors, the variable q_{inlet} can only be observed indirectly. This was accomplished by choosing steady state operational conditions for which the derivative of the manifold pressure diminishes and no exhaust gas recirculation occurs. In this case, q_{inlet} becomes equal to the air signal from the air mass meter.

Figure 8.5 shows the resulting map that was obtained by considering about 50 different measurement points. The measurements points have been interpolated. Due to the necessary interpolation, one cannot assume that f_{mech} corresponds exactly to the obtained map. Assuming an accuracy of ± 1 percent seemed, according to our experiments, sufficient to ensure that the true physical behavior is covered. The function f_{mech} is thus described as a three-dimensional volume that encompasses f_{mech}. This volume was sampled into a finite relational representation. It constitutes the ground mechanical behavior constraint type which constrains the possible combinations of n_{engine}, p_{inlet}, and q_{inlet}.

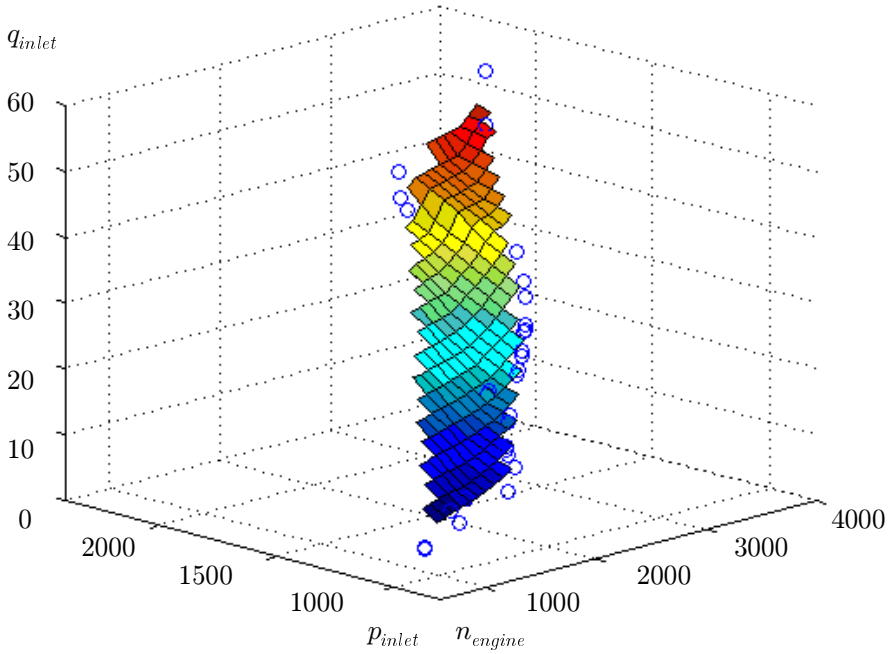

Figure 8.5: Air flow into the engine as a function of engine speed and pressure at the engine intake

Combustion Behavior

The combustion behavioral part of the engine model describes the chemical combustion behavior of the engine in terms of the *stoichiometric ratio* between the combustion ingredients.

The assumption we make here is that factors like the formation of the fuel spray, the geometry of the combustion chamber, etc. are not relevant for the combustion behavior. Following [Kea93, Bau96], the combustion process is then determined solely by the air/fuel ratio, which is denoted λ:

$$\lambda = \frac{actual\ air\ quantity}{theoretical\ air\ requirement}$$

A fuel-lean mixture ($\lambda > 1$) contains more air, while a fuel-rich mixture ($\lambda < 1$) contains less air. The ideal stoichiometric air/fuel ratio is described by

$$\lambda = 1.$$

This condition is often referred to in the literature as "100 percent theoretical air". For diesel fuel, approximately 14.75 kg of air are required to burn 1 kg of fuel completely ([Kea93]), i.e. its theoretical air requirement equals 14.75.

It is important to note that an actual chemical reaction having stoichiometric air/fuel proportions of $\lambda = 1$ will not actually produce complete combustion of the fuel. Since actual oxidation reactions are incomplete, combustion systems have to operate using excess air, i.e. using a fuel-lean mixture. Diesel engines can operate on about 20 percent to 80 percent excess air, corresponding to $\lambda = 1.2 \ldots 1.8$ ([Bau96]). Incomplete combustion and black smoke begins to occur when the excess air is less than 40 percent, corresponding to $\lambda < 1.4$. In the following, this threshold will be referred to as $\lambda_{critical}$. The normal combustion behavior of the engine is then described by the following constraint on the inlet air mass flow and the injected fuel mass:

$$\frac{q_{inlet}}{m_{fuel}} = 14.75 \cdot \lambda.$$

Performance Behavior

The performance behavioral part of the engine model describes the performance behavior of the engine in terms of the produced combustion outputs.

Again, we make the assumption that variations of the fuel spray etc. are not relevant. Due to the assumption of iso-thermic conditions, the combustion products have to be mapped back to the same temperature level as the combustion ingredients. The performance behavior of the engine describes a relationship between λ, inlet air flow q_{inlet} and outlet air flow q_{outlet}. According to [Bau96], the exhaust gas temperature is greater than 800 Kelvin during normal operation of the engine. By applying the ideal gas law (see above) to map the results back to iso-thermic conditions, it follows that the outlet flow is at least 50 percent higher

than the inlet flow. Thus, the following constraint captures the performance behavior of the engine:

$$\lambda \geq \lambda_{critical} \Rightarrow 0.66 \cdot q_{outlet} \geq 14.75 \cdot m_{fuel}.$$

8.2.2 Intake Manifold Model

The intake manifold is the volume between the engine inlet and the turbocharger turbine outlet. It constrains the relationship between inlet pressure p_{inlet}, inlet air mass flow q_{inlet} and turbine air mass flow $q_{turbine}$. The behavior of the intake manifold obeys the ideal gas law:

$$p_{inlet} \cdot V_{manifold} = m_{air} \cdot R \cdot T$$

Because T is constant, it follows for the behavior description of the intake manifold that

$$\dot{p}_{inlet} = \frac{(q_{turbine} - q_{inlet}) \cdot R \cdot T}{V_{manifold}}$$

For the demonstrator car, the parameter $V_{manifold}$ lies between 0.03 and 0.04 m^3.

8.2.3 Turbine Model

The intake turbine of the turbocharger compresses the intake air of the engine, while the exhaust turbine of the turbocharger is driven by the exhaust gas from the engine. The behavior description of a turbine relates its rotational speed to flow and pressure. We first observe that a turbine conserves the mass of air:

$$q_{inlet} = q_{outlet}.$$

Let $Q_{turbine}$ denote the flow rate of the turbine, i.e. the volume that it outputs per one rotation, and $n_{turbine}$ the rotational speed of the turbine. From the ideal gas law, it follows that

$$n_{turbine} = \frac{q_{outlet} \cdot R \cdot T}{Q_{turbine} \cdot p_{outlet}}$$

This relation constitutes the behavior description for a turbine. The parameter $Q_{turbine}$ is known only imprecisely for each turbine and is approximately 0.3 liters. The behavior model of the turbocharger axis states that the rotational speed of the exhaust turbine equals the rotational speed of the intake turbine.

8.2.4 Turbo Control Valve Model

The turbo control valve is opened or closed by the control unit using an electrical duty cycle signal D_{TCV}. No precise parameters were available for this component. Hence, we could use only a coarse model stating that if the ECU commands the turbo control valve to be fully open (corresponding to 90 percent duty cycle), the control pressure $p_{control}$ equals the pressure at the inlet terminal of the TCV, and else it is unknown:

$$D_{TCV} = 0.9 \Rightarrow p_{control} = p_{inlet}$$

8.2.5 Wastegate Valve Model and Converter Model

Like for the turbo control valve, no precise parameters where given for the wastegate valve and the converter that controls it. We only express the knowledge that when the control pressure in the pipe is equal to ambient pressure, the wastegate valve will be fully closed

$$p_{control} = p_{atmosphere} \Rightarrow A_{wastegate} = 0,$$

and that otherwise, the opening area of the wastegate valve is positive:

$$p_{control} > p_{atmosphere} \Rightarrow A_{wastegate} > 0.$$

8.2.6 Control Unit Model

Based on the signals it receives from the sensors, the control unit determines the duty cycle of the turbo control valve and the amount of fuel that will be injected into the engine. Due to the on-board situation, all the signals of the ECU are available to the prototype, which means that this component is fully observed. Diagnosis of this component therefore becomes a problem that is independent of the rest of the device.

Hence, its behavior model can be removed from the DTI model without affecting diagnosis of the remaining components. If we make the assumption that the ECU can never be faulty, modeling it is not necessary for consistency-based diagnosis. Note that this is an important difference compared to methods that are based on performing simulation of the model and comparing the obtained results with the observations in order to perform diagnosis. Such approaches would, in contrast, require a model of (at least parts of) the control unit behavior in order to carry out simulation.

8.2.7 Observations

For the DTI system, the following control unit signals are available: airflow meter signal [mg/stroke], boost pressure sensor signal [hPa], engine speed sensor signal [1/min], injected fuel mass [mg/stroke] and TCV control signal [percent duty].

The atmospheric pressure signal is not considered, as atmospheric pressure is assumed to be constant in the model. The ECU reads in all signals from the sensors simultaneously, i.e. an observation corresponds to a vector at a point in time. The frequency at which the control unit reads in the signals varies with the speed of the engine, therefore the observation vectors are not evenly distributed over time. The on-board diagnosis prototype uses only these signals, i.e. it is based on the same set of signals as the ordinary diagnostic procedures of the ECU.

Note that the sensor signals come in units that are different from the variables used in the model. For instance, airflow is measured in terms of mass per stroke of the engine instead of mass per unit of time. Thus, it depends on the speed of the engine. Therefore, it is necessary to perform preprocessing of the

signals. For instance, the airflow signal has to be transformed to grams per second. Similarly, the injection signal (fuel mass per stroke) combines fuel mass per time and engine speed. It requires a transformation to [g/s]. In the considered system, one revolution corresponds to one stroke of the engine. Thus, in AQUA's signal transformation component, the following conversion is defined between the airflow meter signal (denoted `armM_List`), the engine speed sensor signal (denoted `dzmNmit`), and the respective model variable (denoted `Airflow Sensor.TV.Signal` in the system description):

```
<OBSERVATION>
   <AGGREGATECHANNEL NAME="Airflow_per_sec" OPERATOR="MULT">
      <CHANNEL NAME="armM_List" FILTER="Id"/>
      <CHANNEL NAME="dzmNmit" FILTER="Id"/>
      <CONST VALUE="0.00001667"/>
   </AGGREGATECHANNEL>
   <VARIABLE NAME="Airflow Sensor.TV.Signal"
    DOMAIN="AirflowDom" DOMAINLIBRARY="none"/>
</OBSERVATION>
```

Similarly, the following conversion is defined for m_{fuel}, which is denoted `Engine.TV.Injection.Mass` in the system description:

```
<OBSERVATION>
   <AGGREGATECHANNEL NAME="Injection_per_sec" OPERATOR="MULT">
      <CHANNEL NAME="mrmM_EAKT" FILTER="Id"/>
      <CHANNEL NAME="dzmNmit" FILTER="Id"/>
      <CONST VALUE="0.000016.67"/>
   </AGGREGATECHANNEL>
   <VARIABLE NAME="Engine.TV.Injection.Mass"
    DOMAIN="FuelMassDom" DOMAINLIBRARY="none"/>
</OBSERVATION>
```

It might seem as if this preprocessing of observations assumed correctness of the engine speed signal. However, it does not, as the engine speed signal is just canceled out (later it will be seen that the speed sensor can also be involved in diagnostic hypotheses). In fact, the preprocessing step becomes necessary because the control unit software implicitly assumes correctness of sensors (the speed sensor, in this case). In terms of section 2.2.2, the original airflow meter signal and the fuel mass signal of the ECU are so-called "virtual sensor" signals. However, for model-based problem solving, we need to make the underlying assumptions explicit, and thus we have to determine the respective physical signals. Technically speaking, the above signal transformation amounts to "breaking up" a virtual sensor. The necessity for this illustrates that the current ECU software is still oriented towards traditional approaches to monitoring and diagnosis, and does not follow the concepts of model-based problem solving (see chapter 3).

Signal preprocessing is also necessary to derive the first derivative of the pressure signal, $\dot{p}_{inlet}$. This step is feasible, as there is relatively few noise on the pressure signal compared e.g. to the airflow signal or the engine speed signal. To this end, following the approach of [Nyb99], p_{inlet} is filtered with a low-pass filter with a cut-off frequency of 2 Hz to eliminate the effect of in-cycle variations.

8.2.8 Properties of the Model

Before we proceed with the generation of a qualitative model for the DTI system, we will outline some properties of the behavior models as presented in the previous sections and put them into perspective with other approaches to automotive diagnosis, in particular the work of [Nyb99]. First, the model is *compositional* in the sense that it is organized as a set of behavior fragments of components that do not presume the presence of other components in the device. For instance, the engine model — though its parameters are specific for the Volvo car — does not presume the presence of a turbocharger. This an important difference to the approach taken in [Nyb99], which uses equations that describe the combined behavior of several components.

Second, the model is *incomplete* in that it does not contain precise numerical values for each of its parameters. Many of the parameters in the system are not known precisely, and the best type of knowledge available are ranges, such as for the volume of the intake manifold, or the diameter of the wastegate valve. [NN98, Nyb99] also use a characteristic map in order to describe the behavior of the engine. However, the principled difference is that it corresponds to a numerical function, and not a relation (constraint) as presented in section 8.2.1. Therefore, the model described in [NN98, Nyb99] provides an approximation, rather than an abstraction, of the engine's physical behavior.

A third difference compared to [Nyb99] is that Nyberg focuses on the behavior of the engine, using only an engine testbed instead of a complete car for experimentation. Clearly, an actual demonstrator car bears more complexity in terms of the different driving situations that can occur. Of course, the applicability of our model to the real vehicle is still limited due to the modeling assumptions which might by violated in real driving situations, particularly e.g. in the case of exhaust gas re-circulation.

8.3 Generating a System Description

The key to provide the efficiency that enables to run model-based diagnosis in an on-board environment is to avoid the computational complexity of numerical modeling and simulation. The theoretical background and the software components that have been developed in this thesis can be used in order to support qualitative modeling of the DTI system. AQUA can be helpful in two respects:

(1) to turn the (real-valued) model to a ground system description, i.e. to generate finite constraints for the behavior descriptions of the components in the DTI model;

Quantity	Domain
Pressure p	$\{[0, 1610], (1610, 1770), [1770, 1860], (1860, \infty)\}$
Air flow q	$\{(-\infty, 31.0], (31.0, 37.5), [37.5, 42.5], (42.5, \infty)\}$
Fuel mass m	$\{(-\infty, 1.53), [1.53, 1.68], (1.68, \infty)\}$
Engine speed n	$\{(-\infty, 2490), [2490, 2690], (2690, \infty)\}$

Table 8.2: Ground domains for variables

(2) to determine a level of granularity for the DTI model that is adequate for the specific task we are after, namely diagnosing failures related to incomplete combustion.

In the following, we describe results that have been achieved using the prototypic implementation of AQUA as described in chapter 7 and a Windows PC running NT 4.0 with AMD Athlon 700 MHz CPU and 128 MB Ram.

8.3.1 Ground Model

Using the model fragments outlined in section 8.2, a behavior model of the DTI system has been composed within the Raz'r development system. The resulting system description consists of 16 components, 146 variables and 60 constraints.

The parameters for the model and the characteristic map for the engine are specified in a real-valued parameter description. The domains of the variables have first to be turned to finite domains in order to make them amenable to further model-based reasoning. As black smoke occurs only during high power demands, we use domains that concentrate around typical values in full duty. The maximum fuel consumption is about 1.70 grams per second, the maximum airflow is about 45 grams per second, while the maximum boost pressure lies around 2000 hPa. The domain for variables involving pressure, air flow, fuel mass and engine speed were chosen as shown in table 8.2.

The domain for the derivative of pressure distinguishes values less than -10 from values greater than 10, and the domain for the turbo control valve duty cycle distinguishes full duty (corresponding to 90 percent) from the rest of the values. The domains for parameters in the model distinguish the parameter values from the remaining real values.

Based on this, the System Description Generator component of AQUA can be used to generate an initial model of the DTI system. The generation of the respective constraint types takes about 2 seconds. The resulting system description for the DTI has a size of approximately 100 KBytes.

8.3.2 Diagnostic Results

Using AQUA's Signal Transformation component, the derived model can be used for diagnosis using real measurements from the demonstrator car. In the following, an example for a measurement of the Volvo demonstrator car is presented that has been taken for the first scenario, which involved a leakage in the air

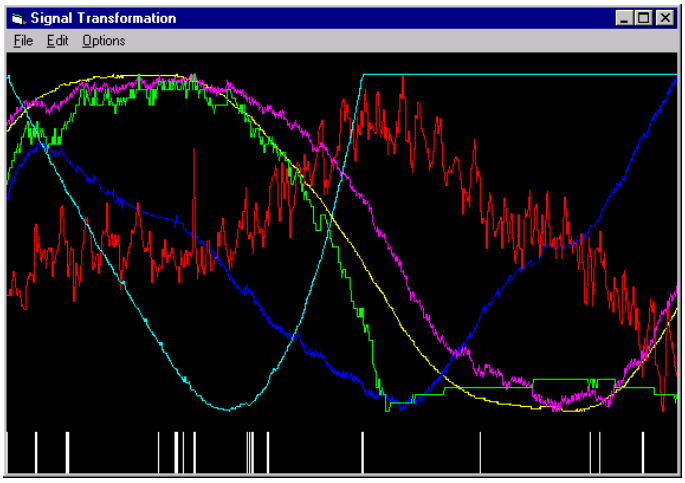

Figure 8.6: AQUA's Signal Transformation Component showing measurements for leakage scenario (the bars at the bottom of the window indicate the generated qualitative observation vectors)

intake manifold. Figure 8.6 shows a screenshot of the Signal Transformation component for this measurement file.

The measurement runs for 9.75 seconds. The leakage is opened about 3 seconds after starting the measurement, and is closed again after approximately 8 seconds. The acquired measurement file consists of 1053 quantitative observation vectors. Using the initial domain abstraction, the signal transformation component generates 28 qualitative vectors at time points t_1 to t_{28}. The run-time for signal transformation is less than 1 second. The resulting observation file and the system description can be fed into Raz'r's run-time diagnosis system. Figure 8.7 shows a screenshot of the run-time system with the DTI model loaded.

The run-time system derives the following results. For the qualitative observation vector t_{19}, which corresponds to real-time 3.84 seconds, a first conflict is detected:

{`Airflow Sensor.ok, Junction1.ok, Intake Turbine.ok, Junction2.ok, Junction3.ok, Manifold.ok, Pressure Sensor.ok, Engine.ok, Speed Sensor.ok`}

The physical explanation for this conflict is as follows. At t_{19}, the pressure sensor signal drops considerably, though remaining still within the normal region. The airflow sensor signal remains unchanged. From the model of the intake manifold, it follows that the air flow into the engine must be higher. Because the speed sensor signal remains unchanged, it follows from the engine model (mechanical behavior) that the boost pressure must have become considerably higher. However, the pressure sensor signal is still normal, which yields a discrepancy. That is, the conflict arises because the pressure signal suddenly drops, but the airflow

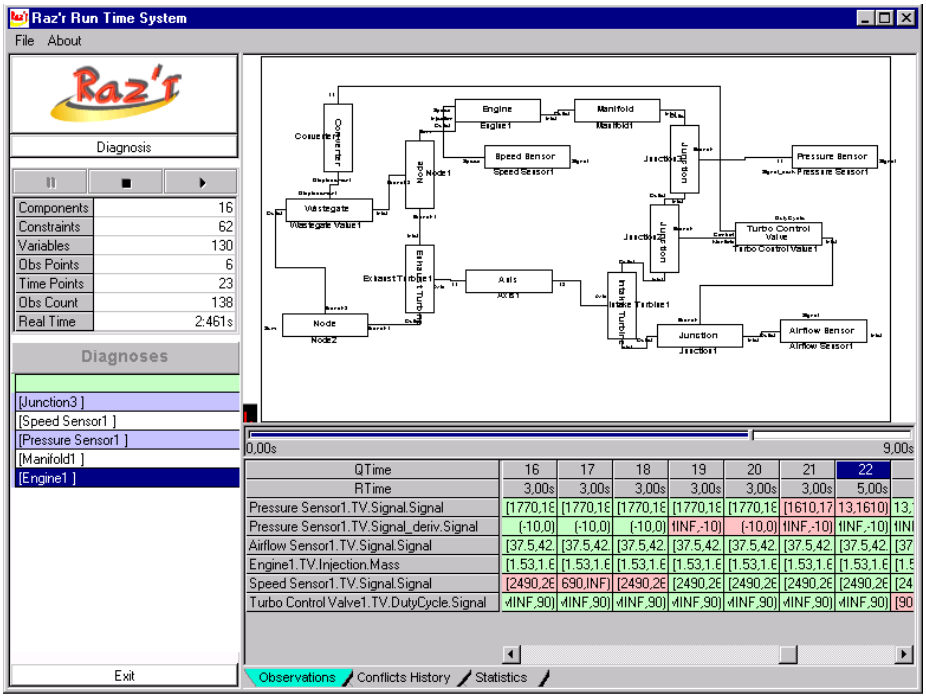

Figure 8.7: Screenshot of the run-time system for the leakage scenario

signal and the speed signal remain at their previous levels, which is inconsistent with normal behavior. The boost pressure signal drops because due to the leakage in the manifold component, air is beginning to escape into the atmosphere. The corresponding diagnosis consists solely of single faults:

{Airflow Sensor.ok}
{Engine.ok}
{Junction3.ok}
{Speed Sensor.ok}
{Junction2.ok}
{Junction1.ok}
{Pressure Sensor.ok}
{Manifold.ok}
{Intake Turbine.ok}

At t_{22}, which corresponds to real time 5.20 seconds, a second, smaller conflict is detected:

{Engine.ok, Junction3.ok, Speed Sensor.ok, Pressure Sensor.ok, Manifold.ok}

This can be explained as follows. At t_{22}, the pressure sensor signal has become low. However, the speed signal is still at its normal level, and also the amount of

fuel injected remains unchanged. From the engine model (combustion behavior), it follows that the minimum airflow required for complete combustion of the fuel is still at a certain high level. However, from the engine model (mechanical behavior) it follows that due to the reduced pressure, the actual airflow must be lower than this level. That is, the conflict arises because the boost pressure is too low for the given engine speed and amount of fuel injected. The boost pressure is low because due to the leak in the manifold, the intake manifold cannot build up the required high pressure. The corresponding diagnosis which combines the first and the second conflict consists of a smaller number of single faults than the previous one:

{Engine.ok}
{Junction3.ok}
{Speed Sensor.ok}
{Pressure Sensor.ok}
{Manifold.ok}

Finally, at t_{23}, which corresponds to real time 5.23 seconds, two large conflicts occur:

{Speed Sensor.ok, Manifold.ok, Axis.ok, Intake Turbine.ok,
Airflow Sensor.ok, Turbo Control Valve.ok, Junction2.ok,
Wastegate Valve.ok, Converter.ok, Junction3.ok, Engine.ok,
Exhaust Turbine.ok, Junction1.ok, Node1.ok, Node2.ok}

{Pressure Sensor.ok, Manifold.ok, Axis.ok, Intake Turbine.ok,
Airflow Sensor.ok, Turbo Control Valve.ok, Junction2.ok,
Wastegate Valve.ok, Converter.ok, Junction3.ok, Engine.ok,
Exhaust Turbine.ok, Junction1.ok, Node1.ok, Node2.ok}

The physical explanation is as follows. At t_{23}, the TCV duty cycle is equal to full duty. From the TCV component it follows that the control pressure is equal to the pressure at the air inlet (i.e. atmospheric pressure). From the converter model and the wastegate valve model it follows that the wastegate valve is fully closed, and that the flow across the exhaust turbine is equal to the airflow from the engine outlet. From the engine model (combustion behavior) and the injection signal being still unchanged it follows that the airflow into the engine must be at least at a sufficient level, and from the engine model (performance behavior), a certain minimum airflow from the engine outlet can be derived. From the exhaust turbine model, it follows that the rotational speed of the axis is relatively high. Thus, from the intake turbine model and the airflow signal being normal, it follows that the boost pressure must be relatively low. This conflicts both with the measured boost pressure and the minimum boost pressure derived from the engine model (mechanical behavior) and the measured engine speed.

That is, for the observations at this time point, the airflow across the turbocharger intake turbine is too low for the amount of fuel injected and the wastegate valve being fully closed. What actually happens in this situation is that the

ECU commands the TCV to rise boost pressure (without noticing the failure, though), which is not successful due to the leak in the intake manifold. The final diagnosis which combines the three conflicts consists of three single faults and a number of double faults:

```
{Engine.ok}
{Junction3.ok}
{Manifold.ok}
{Speed Sensor.ok, Node2.ok}
{Speed Sensor.ok, Node1.ok}
{Speed Sensor.ok, Exhaust Turbine.ok}
{Speed Sensor.ok, Converter.ok}
{Speed Sensor.ok, Wastegate Valve.ok}
{Speed Sensor.ok, Turbo Control Valve.ok}
{Speed Sensor.ok, Intake Turbine.ok}
{Speed Sensor.ok, Airflow Sensor.ok}
{Speed Sensor.ok, Junction2.ok}
{Speed Sensor.ok, Junction1.ok}
{Speed Sensor.ok, Axis.ok}
{Pressure Sensor.ok, Node2.ok}
{Pressure Sensor.ok, Node1.ok}
{Pressure Sensor.ok, Exhaust Turbine.ok}
{Pressure Sensor.ok, Converter.ok}
{Pressure Sensor.ok, Wastegate Valve.ok}
{Pressure Sensor.ok, Turbo Control Valve.ok}
{Pressure Sensor.ok, Intake Turbine.ok}
{Pressure Sensor.ok, Airflow Sensor.ok}
{Pressure Sensor.ok, Junction2.ok}
{Pressure Sensor.ok, Junction1.ok}
{Pressure Sensor.ok, Axis.ok}
{Pressure Sensor.ok, Speed Sensor.ok}
```

The run-time of the RTS is 2.79 seconds. This means that for this example, the performance of the on-board prototype is in the order of magnitude of real-time. Similar results where achieved for the rest of the scenarios. Table 8.3.2 summarizes the results. Scenario 5 was found to have no effects for the considered driving situations. Note that the current control unit software, based on the same signals, is not able to detect any of the above failures. Because some failure effects are noticeable only during certain operating conditions, the diagnosis system cannot always determine a unique diagnosis, but rather yields a number of hypotheses as in the example above. E.g. for the scenario with the boost pressure sensor out of tune, the diagnosis system yields two conflicts and outputs a list of three single faults which contain the boost pressure sensor as one possible candidate, but also other components that together could account for the same symptoms. In these cases, knowledge about the behavior of faulty components,

Scenario	Fault detec.	Single fault hyp.	Real-time	Quantitative obs.	Qualitative obs.	Run-time
1	yes	3	9.75 s	1053	28	2.79 s
2	yes	8	20.21 s	1364	4	1.55 s
3	yes	8	25.35 s	1711	23	2.63 s
4	no	–	14.48 s	1870	42	3.12 s
5	no	–	7.95 s	537	1	1.33 s

Table 8.3: Diagnostic results for the DTI model

i.e. fault models, could be used to further constrain the set of diagnostic candidates. So far, only models of correct behavior have been used for the diagnostic experiments, because the current version of the run-time system cannot deal with fault models. However, at least in some cases, there is evidence that fault models could be useful to partially compensate for the limited observability and, thus, to further restrict the diagnostic candidates.

8.4 Task-dependent Qualitative Abstraction

Providing the efficiency to run model-based diagnosis on-board requires to consider only essential distinctions in the model. In the context of on-board diagnosis, using models that have a maximal, but still adequate level of granularity is beneficial in two respects. First, the size of the model itself is reduced in terms of the size of the involved constraints, hence its time and space requirements are smaller. Second, fewer inputs in terms of observation vectors have to be considered, as the number of observations that are qualitatively different is decreased. Both these effects are instrumental to meet the requirements of on-board diagnosis, in particular to provide the required response times. In this section, we describe how AQUA can be used to derive a task-dependent model of the DTI system.

8.4.1 Target Distinctions

Our goal is to distinguish the situations where combustion is incomplete — and therefore black smoke occurs — from the normal situations where combustion is complete. The latter situation could make it necessary for the ECU to take an action in order to inform the driver (see [Cod93]). As noted in section 8.2.1, the combustion process is determined by λ and the critical threshold $\lambda_{critical}$. In our framework, this modeling goal can thus be expressed as a target partition for λ, stating whether it is above or below the critical value $\lambda_{critical}$:

$$\pi_{targ,\lambda} = \{(-\infty, 1.4), [1.4, \infty)\}.$$

8.4.2 Observable Distinctions

The fact that only certain variables in the DTI system are measured can be expressed as an observable distinction for the variables in the system description. It associates the identical domain mapping with variables that correspond to ECU signals listed in section 8.2.7, and the trivial domain mapping with all the other variables.

8.4.3 Transformed Model

We can then use AQUA's component for Computation of Induced Distinctions to derive induced distinctions for the DTI model. Building the SD Tree for the system description of the DTI model and minimizing it takes approximately 12 seconds. The SD Tree contains 31 meta-variables. After minimization, the metavariable corresponding to the engine component, for instance, has 1732 consistent tuples out of a tuple space of 207360.

The largest intermediate metavariable in the SD Tree has 11 variables in its scheme, i.e. less than 8 percent of all variables. The largest intermediate metavariable has also less variables than the largest component model, which means that during the construction of the SD Tree, no intermediate results occur that would exceed the size of a component model.

We first run AQUA applying the incomplete condition captured in theorem 2. It turns out that this yields no distinctions for the variables. This implies that it is only the *combination* of external restrictions for different variables that can derive different solutions on the level of target distinctions, and we have to use the complete condition (theorem 3) in order to derive induced distinctions.

Applying theorem 3 requires as a precondition that the set of external restrictions is consistent with the model.

AQUA determines that for the DTI model and the observable distinctions corresponding to all sensor signals, not all of the external restrictions are consistent with the model. This is due to the fact that there exist observations that conflict with the model of a correct manifold.

As a result, we either have to change the observable distinctions or augment our system description with a fault model of the manifold (see section 3.6.3). As the RTS currently cannot process fault models, we choose to limit ourselves to a subset of observations that readily fulfills the precondition for applying theorem 3 with the given model.

Consider the subset of observations consisting of the boost pressure sensor signal (without its derivative), the mass airflow sensor signal, the engine speed sensor signal and the fuel quantity injected.

With this reduced set of observations, of course, we cannot expect to get all the conflicts as with the ground model. Given the respective observable distinctions and the target distinctions for λ as described above, it takes AQUA about 2 minutes to compute the solution partition $\Sigma(R, \boldsymbol{\tau_{obs}}, \boldsymbol{\tau_{targ}})$ on the SD Tree. AQUA derives the following induced partitions:

```
<Partition VARIABLE_NAME="Pressure Sensor1.TV.Signal.Signal">
   <PartitionElement>
      <Value VALUE="[0,1610]"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="(1610,1770)"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="[1770,1860]"/>
      <Value VALUE="(1860,INF)"/>
   </PartitionElement>
</Partition>

<Partition VARIABLE_NAME="Speed Sensor1.TV.Signal.Signal">
   <PartitionElement>
     <Value VALUE="(MINF,2490)"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="[2490,2690]"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="(2690,INF)"/>
   </PartitionElement>
</Partition>

<Partition VARIABLE_NAME="Engine1.TV.Injection.Mass">
   <PartitionElement>
      <Value VALUE="(MINF,1.53)"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="[1.53,1.68]"/>
   </PartitionElement>
   <PartitionElement>
      <Value VALUE="(1.68,INF)"/>
   </PartitionElement>
</Partition>

<Partition VARIABLE_NAME="Airflow Sensor1.TV.Signal.Signal">
   <PartitionElement>
      <Value VALUE="(MINF,31.0]"/>
      <Value VALUE="(31.0,37.5)"/>
      <Value VALUE="[37.5,42.5]"/>
      <Value VALUE="(42.5,INF)"/>
   </PartitionElement>
</Partition>
```

That is, AQUA determines that it is unnecessary to distinguish between values for pressure that are equal to or above 1770 hPa, and that distinguishing values for the airflow is not useful. The latter is due to the fact that because the derivative of the pressure signal and the TCV command are not observed, the airflow signal cannot be used to determine the airflow into the engine. AQUA also outputs a domain mapping from the real numbers to the induced domain values that can in turn be used for generating the corresponding abstracted system description and for signal transformation of the measurements.

8.4.4 Diagnostic Results

For the same test case as presented in section 8.3.2, the signal transformation component generates 12 qualitative observation vectors at t_1 to t_{12}. Using these observations and the transformed system description, the Raz'r run-time system finds a conflict for t_{12}, corresponding to real time 5.20 seconds:

{`Engine.ok, Junction3.ok, Speed Sensor.ok, Pressure Sensor.ok, Manifold.ok`}

The conflict is the same as the one that has been found at t_{22} for the original model of the DTI. The computing time required for the run-time system is 1.84 seconds as compared to 2.79 seconds for the original model.

8.5 Evaluation and Discussion

The demonstrator described in this chapter illustrates the feasibility of automated qualitative modeling in the automotive domain. Besides the results for the diagnostic scenarios, also the way in which they have been achieved is important. Note that the device under consideration, like many other automotive systems, comprises standard physical components whose behavior can be described in terms of (differential) equations, but also elements like the engine for which no rigorous mathematical model exists. Consequently, the behavior models of the various parts are rather heterogenous, ranging from continuous-valued variables for the physical part to discrete control signals and characteristic lines or maps for components or parts of the system for which no algebraic relationships can be devised.

As has been shown, automated domain abstraction supports the composition and smooth integration of such independently developed, "hybrid" component models, as it allows the modeler to express his knowledge about the physical behavior of components of the system without being committed to a specific abstraction level. It thus allows a defined way of modeling that is "incremental" in the sense that adding more complete knowledge about the behavior of the components leads to more accurate results. In contrast, qualitative models that rely on a fixed abstraction level for the domains of variables (e.g., signs) often face a built-in limit from which on additional knowledge (e.g., about the range of parameters) can't be exploited any more.

From the perspective of the potential users, modeling components and systems by just writing down the equations and specifying the characteristics of the task one is after appears more natural than having to define qualitative constraints. Therefore, we argue that AQUA makes the process of modeling for model-based problem solving more acceptable to engineers, less error-prone, and more cost-effective due to automating steps that up to now had to be done by hand (see also the discussion in chapter 7). Of course, one has to be aware that these conclusions have to be validated and confirmed by further studies.

The approach by Nyberg and Nielsen ([NN98, Nyb99]) to diagnosis of leakages and other faults in automotive systems is based on numerical methods. First of all, it underlines that diagnosis of leaks in the air-intake system is indeed an interesting application problem. Because the system they consider comprises a gasoline (i.e. spark-ignition) engine instead of a diesel engine and includes different components such as a throttle, the diagnostic results they report are not directly comparable with ours.

[NN98, Nyb99] use the behavior models within a diagnosis framework that is based on evaluating residuals. Compared to the consistency-based approach to model-based diagnosis as outlined in chapter 3, there is no explicit notion of the structure of the system; instead, it is implicitly coded within the residuals, which have to be determined manually. A problem of this approach is that whenever the structure of the model is changed (e.g., because additional components are added), the residuals have to be determined again by hand.

To perform this step, however, seems particularly difficult for the type of system considered in this chapter. Note that for the DTI, we could not derive induced distinctions by considering the incomplete condition of theorem 2. The interpretation for this is that if we want to determine — based on the sensor signals as external restrictions — whether black smoke occurs or not, we have to consider external restrictions that involve several variables at once. Hence, the system, though it consists of a rather small set of components only, bears a difficult analytic redundancy that goes beyond the separated evaluation of sensor signals. This makes the determination of appropriate residuals complicated, and is probably the main reason why the current on-board diagnostics of DTI cannot diagnose the faults that have been installed in the demonstrator car. Instead, model-based reasoning constitutes the foundation to do this systematically and automatically.

Note also that explaining the results in terms of the annotations given in section 8.3.2 is only made possible through an approach that breaks down the behavior of the system into behavior of individual components. In a sense, the results of task-dependent abstraction can be viewed as providing a "semantics" for informal notions such as "normal", "low", "significantly higher", etc. that have been used to explain the physical behavior that lies behind each of the conflicts.

Other possible applications of task-dependent automated qualitative abstraction to the DTI model would involve the definition of failure modes and recovery actions. For example, if a failure model for the manifold component was defined

that describes a leakage fault, we could specify a target distinction (i.e. threshold) for the size of the leak. AQUA could then be used to derive a model whose granularity is suited for diagnosing leakages of at least this size, while neglecting smaller leakages.

Finally, an important application is to find a granularity of the model that is tailored to the available recovery actions of the control unit. If we had a list of the possible recovery actions together with their severity, a model could be derived that has the necessary granularity for the ECU in order to decide which recovery action to take. Actually, a subgroup within the VMBD project has experimented with the automated selection of recovery actions based on a model of the diesel injection subsystem ([CCG+99]).

8.6 Summary

This section presented a model-based system that diagnoses problems related to increased carbon emissions of diesel engines, a problem of significant importance with respect to environmental impact and compliance with legal requirements. The prototype transforms the sensor signals that are available to the standard electronic control unit to a qualitative level and exploits them for detecting and localizing faults based on a model of the system. It has been evaluated on a Volvo demonstrator vehicle with a number of built-in faults.

The prototype illustrates the feasibility and the benefits of automated qualitative abstraction in the automotive domain. It can provide a basis for a systematic and cost-effective approach to creating diagnostics for car subsystems, and has the potential to improve the quality of diagnostics by handling fault situations that are not covered by current state-of-the-art on-board diagnostics. At the same time, the outcomes represent a major step in the transfer of model-based reasoning techniques to the automotive industry.

Chapter 9

Summary

This thesis deals with the general and important question: What to include in a behavior model, and what to leave out? This question constitutes a core problem of modeling physical systems, and it is of particular interest how answers to this problem can be provided automatically.

To this end, we investigated a more specific subproblem: What distinctions have to be included in the domains of variables of a behavior model, and what distinctions can be abstracted away without affecting the result one is interested in? This question is equal to finding qualitative values for the domains of variables in a model that are adequate for a specific problem-solving task.

Hence, in a first part, we provided an explicit notion of task-dependency in order to reason directly about such aspects as observable and desired distinctions, necessary distinctions, and unnecessary distinctions in a model. The ability to explicitly reason about task-dependency constitutes the prerequisite to automatically finding qualitative abstractions of a model. The problem could be formalized as finding so-called induced distinctions within the domains of variables that are both necessary and sufficient, given a system description composed from a library, a granularity of possible observations, and a granularity of the desired results. We presented fundamental results regarding solutions to task-dependent qualitative abstraction, and identified relationships between qualitative reasoning and constraint satisfaction techniques for structuring problems and compactly describing their solutions.

The methods devised for computing qualitative domain abstractions are based on the SD Tree as an implicit, hierarchical representation of the possible behaviors, which allows to exploit the specific structure of a device model in order to avoid combinatorial explosion. Thus, the computational complexity of deriving induced distinctions could be bound to structural properties of the system description. The resulting methods have been implemented in a prototypic system called AQUA (Automated Qualitative Abstraction). AQUA's software components build on an existing model-based reasoning framework that allows to define domains, constraint types and device structures for composing system descriptions and to use them for performing behavior prediction and diagnosis. AQUA automates the transformation of models to a level of abstraction adequate for

a specific structure and task, much like an engineer's ability to come up with a suitable representation when faced with a certain problem. Thus, AQUA supports several tasks in the context of model-based problem solving that up to now essentially had to be carried out manually.

In a second part, we illustrated this using examples taken from the automotive domain. One principled application is to turn real-valued models, as commonly used in industry, into qualitative models to make them accessible to model-based reasoning methods. As a major result, a prototype of a model-based on-board diagnosis system for a passenger vehicle has been presented. The prototype is shown to provide useful results for a number of emission-related failure scenarios that were implemented on a Volvo demonstrator car. This demonstrates that AQUA can greatly enhance the ability to use a behavior model of an engineered device as a common basis to automatically support different tasks along its life cycle.

9.1 Related Work

Approaches that are related to specific aspects of our work have been described in the discussion sections of respective chapters. In this section, we highlight differences to existing work that emerge from a broader, conceptual perspective.

Since qualitative modeling is important from an application point of view, a considerable amount of literature deals with this topic (see [WdK90, FS92] and section 4.5 in chapter 4). However, *automating* qualitative modeling rises many scientific challenges, and there hardly exist any software systems that can be readily used in the context of application problems. Research on automated modeling has primarily focused on the question of how to compose a model from a set of model fragments, and how to decide which model fragments have to be included in the model.

Nayak ([Nay95], see also section 4.5.2) deals with the problem of automatically composing an adequate model of a device by selecting appropriate model fragments from a library. The specific problem-solving task pursued is to obtain parsimonious causal explanations of device behavior, which can be formulated as a user query about the interaction of certain variables in the model. The model selection algorithm devised in [Nay95] ensures that a causal explanation can still be derived from the resulting model, i.e. the model is suitable for the pursued task, while it cannot be guaranteed that the obtained model indeed leads to a most parsimonious causal explanation, i.e. the model is not necessarily "optimal" with respect to the pursued task. Nayak deals with model fragments formulated in terms of (differential) equations over the domain of real numbers, and is not concerned with domain abstraction. In fact, domain abstraction does not combine in a straightforward way with the underlying technique called causal ordering, because due to the involved ambiguities, causal ordering will not work in an analogous manner for qualitative constraints. Nayak assumes that model fragments that can be combined to capture the essential aspects of behavior are

readily available in a library. In contrast, our method *transforms* model fragments, hence the search space is different, consisting of possible transformations of model fragments instead of possible selections of model fragments. A further difference is that the notion of the task pursued in [Nay95] is of limited use regarding the aim to support basic problem-solving tasks of behavior prediction and diagnosis that are pursued in this thesis. The fundamental reason is that in [Nay95], the resulting model not necessarily covers the same physical behaviors as the base model, i.e. it is an approximation rather than an abstraction. In contrast, AQUA's domain abstractions are sound abstractions, guaranteeing that the result covers the same physical situations as the base model. This turns out to be an important precondition for problem solving based on the model.

Like AQUA, Williams' MINIMA system ([Wil91], see section 4.5.5) also captures the idea of obtaining — starting from a base model — optimal information with respect to a targeted level of distinctions. In MINIMA, the targeted distinctions are built-in and correspond to the signs of the variables, whereas the base model is assumed to be real-valued. However, there is no notion that would correspond to observable granularity. The core of MINIMA is a set of symbolic transformation rules that allow to simplify and factorize algebraic expressions composed of reals and a domain abstraction operator for signs, such that the information about the sign of the result will be preserved as far as possible. MINIMA can be run both in an incomplete mode where completeness of the results is traded for faster, local deductions, and a complete mode which comes at the expense of potentially generating intractably large intermediate expressions. Unlike MINIMA, the approach we presented is not limited to a fixed granularity of the results corresponding to sign abstraction, but instead allows to specify target distinctions in an arbitrary and explicit way. AQUA is also more general as it is not restricted to algebraic constraints such as addition or multiplication. E.g., in section 8.2.1, a constraint describing the mechanical behavior of a diesel engine was described as a characteristic map. On the other hand, deriving induced distinctions with AQUA requires a finite base granularity to start from, and cannot exploit algebraic relationships that would sometimes allow to draw "obvious" results, such as for equality. Compared to MINIMA, qualitative abstraction of real-valued models with AQUA more or less amounts to a "brute-force" approach of first generating finite constraint types, regardless of possible symbolic simplifications, and then discarding any distinctions that turn out unnecessary.

QSIM ([Kui94], see also section 4.3.2) is a system for performing qualitative simulation of device behavior over time, based on (incomplete) information about initial magnitudes and directions of the variables in a device model composed of qualitative differential equations (QDEs). QSIM incorporates methods for refining the domains of variables, i.e. deriving new landmarks and corresponding values, during simulation. However, except for signs, the mapping of the qualitative values to their base domain remains unknown. Instead, only information on their ordinal relationship is provided, which is quite weak and for instance not sufficient to simplify the constraints of the behavior model or to abstract real-valued measurements as in AQUA. Q2 and Q3 ([Kui94, BK92]) are exten-

sions of QSIM that addresses these shortcomings and allow to perform so-called semi-quantitative reasoning. The idea is to derive from the QDEs and qualitative behaviors generated by QSIM algebraic constraints that involve the landmarks as variables. The resulting CSP can then be solved using interval propagation to derive numeric bounds on the landmark values. However, the techniques are intrinsic to the specific context of simulating behavior over time. Another important difference to our work is that, as noted above, AQUA allows to process arbitrary relations, and is not limited to pre-defined algebraic constraint types or monotonic functions that can be represented as QDEs.

The thesis also revealed interesting relationships between qualitative reasoning and abstraction techniques originating in constraint satisfaction, notably Freuder's work on interchangeability (see [Fre91, FS95, WF99] and section 4.6.1). Weigel and Faltings ([WF99]) analyze the theoretical limits of interchangeability in constraint satisfaction problems. The intuition is that the larger the set of possible solutions to a problem is, the larger should be the likelihood for interchangeability to occur. Accordingly, a combinatorial increase in the number of tuples of a relation should be compensated for by increasing possibilities to represent these tuples more compactly. Weigel and Faltings provide theoretical foundations for this intuition by means of the theory of error-correcting codes, and give a bound for the number of solutions of a CSP that can exist without being interchangeable. Since interchangeability corresponds to a special case of induced distinctions for diagnosis (section 5.6), it follows that the required granularity of a model used for diagnosis tends to decrease with the number of tuples that the model relation permits. However, the analysis in [WF99] emanates from random constraint satisfaction problems and does not take into account the specifity of model relations describing physical devices.

9.2 Conclusions

9.2.1 Achievements from a Scientific Point of View

Qualitative reasoning is an active area of research. People who are less familiar with the field sometimes find it difficult to identify a common "thread" as there are many different directions pursued ([WdK90]). One reason for this is that the notion of qualitative values and qualitative models is inseparably related to the task to be solved. As our fist contribution, the notion of a qualitative abstraction problem (QAP) captures the essence of task-dependent qualitative abstraction within a common relational framework, formalizing the problem and making it amenable to further theoretical analysis. Our second contribution is to derive, based on this formalization, a first-principles solution to task-dependent qualitative model abstraction in the form of induced distinctions and to characterize various interesting properties of induced distinctions. Third, algorithms and data structures have been devised that can be used as a basis for efficient computation of solutions to the problem on a computer. Based on this, we were finally able to identify principled applications of automated qualitative abstraction, such as de-

riving suitable distinctions for model-based diagnosis. Overall, we have provided a contribution to deriving a logic-level description of a problem from a physical level in a systematic way, which in [Wal99] has been identified as one of the main challenges in AI research.

9.2.2 Achievements from an Application Point of View

The increasing complexity of engineered devices, particularly — but not exclusively — in the domain of automotive systems, leads to an increased demand for computer-supported behavior prediction, diagnosis, and testing. Given the increasing maturity and scale of model-based systems applications, the question of how to re-use model fragments and to compose models from generic libraries is of growing industrial interest.

In an ideal work process, an engineer would be able to specify a (real-valued) ground behavior model, provide information about the goals and conditions of the pursued task, and let an adequate model be generated automatically based on this knowledge. AQUA represents a step towards this ideal. It implements methods that enable to automatically transform a behavior model to a granularity adequate for the task at hand, thus supporting the re-use of knowledge along various tasks. Hence, AQUA can be seen as a contribution to bridging the gap between a traditional engineer's way of thinking (which is mostly oriented towards quantitative methods), and AI techniques (which have put forth qualitative abstraction), a problem that has been identified as one of the major roadblocks to a more wide-spread industrial application of model-based reasoning methods.

A first application in this direction has been demonstrated in chapter 8, where the output of AQUA was the basis for a prototype of a model-based diagnosis system that runs on-board a real passenger vehicle with built-in faults. This demonstrator provided useful diagnostic results for a set of failure scenarios and exploited the devised methods in two respects: qualitative abstraction of the model reduced the complexity of reasoning with the model, while qualitative abstraction of the available observations reduced the number of points in time at which this reasoning had to be initiated at all. These results are instrumental to meet the stringent run-time and space requirements of on-board diagnosis.

AQUA thus provides a first approach for systematically capturing and exploiting the interrelationships between the granularity of a behavior model and the distinctions that are available or required for the particular task that needs to be solved.

9.3 Future Work

9.3.1 Directions for Scientific Work

It has been mentioned in section 9.1 that AQUA has no notion of algebraic relationships, but instead treats any behavioral information as a relation over some (finite) domain. On the one hand, this captures the most general case and allows

to exploit knowledge that goes beyond pre-defined algebraic constraint types such as addition or multiplication. On the other hand, it deprives AQUA from exploiting algebraic relationships that would sometimes allow to draw "obvious" results which hold independently from the granularity of the involved domains. Hence, an interesting direction would be to combine AQUA with some form of algebraic reasoning, e.g. in the sense that only at a point where the manipulation of algebraic expressions becomes too complex, abstraction to finite relations comes into effect. Currently, the approaches pursued in MINIMA and AQUA might be regarded as two extremes in a spectrum of possible strategies that intertwine algebraic and qualitative reasoning.

AQUA is a solution to the problem of qualitative abstraction that works for arbitrary models specified in terms of relations, provided that the domains of the involved variables are finite. The problem of finding qualitative abstractions of real-valued base models has been approached only indirectly by applying an "initial" domain abstraction that reduces the problem to this case. An important question is therefore whether the methods can also be applied to real-valued constraint types, or at least to restricted cases of real-valued constraints such as monotonic functions or piecewise linear functions. For real-valued functions, partition elements could be represented implicitly by interval borders. Note that for the special case of real-valued monotonic functions and target distinctions that can be expressed as landmarks, deriving induced distinctions becomes equal to the problem of finding corresponding values for these landmarks. However, it is clear that the general case involves problems similar to those occuring in symbolic algebraic manipulation. It would be an interesting challenge to identify problem subspaces for which task-dependent qualitative abstraction with real-valued relation types is feasible. Specifically, an open problem in this context is to indentify conditions under which the application of a domain abstraction to a real-valued constraint can yield possibilities for qualitative abstraction (see example 8 in chapter 5). More work would also be necessary to automate the method for iterative domain abstraction that has been proposed in section 7.2.4.

Another issue concerns the identification of tractable classes of finite qualitative abstraction problems. The theorems in chapter 6, which relate the complexity of finding induced distinctions to the size of the largest meta-variable in a SD Tree, constitute a principled starting point. E.g. in section 6.6, resistive networks have been identified as one "best-case" class of problems for which the size of the meta-variables remains constant. Determining tractable classes of qualitative abstraction problems based on such structural properties of the behavior model is related to the problem of identifying optimal structural decompositions of constraint graphs (see [GLS00]).

The SD Tree as a data structure to efficiently represent a model relation allows to derive qualitative abstractions, but it can also be useful for other tasks involved in model-based problem solving. In particular, it can provide the basis for performing sound and complete behavior prediction and diagnosis. The principle is that external restrictions (i.e. observations) correspond to restrictions of the SD Tree, while sets of components that are inconsistent with each other (i.e.,

that form a conflict) correspond to meta-variables whose domain becomes empty during minimization. [MS99] describes an approach to consistency-based diagnosis that is based on clustering behavior models. In fact, consistency checking using the SD Tree has been applied to verify the diagnostic results presented in chapter 8.

A problem we have not dealt with is the impact of specific solution algorithms on task-dependent model abstraction. In particular, incomplete solution algorithms for model-based problem solving — e.g. based on partial constraint satisfaction methods — may not be able to exploit all derived distinctions, and, consequently, they might offer further possibilities for abstraction of a behavior model. However, since the ultimate reason for solution algorithms being incomplete are efficiency considerations, the question is whether the extra effort required for identifying such situations would be worthwhile.

9.3.2 Directions for Applications

A main direction for future work from an application point of view is the integration of the developed software tools into the current work process of engineers, such as e.g. developers of vehicles. In fact, the results and the experience gained during the VMBD project already provided a basis for positive decisions about introducing model-based and qualitative reasoning technology by the industrial users. A follow-up project again joins a number of car manufacturers, a supplier of vehicle subsystems, a software supplier, and a number of research groups from European universities. It aims explicitly at turning the technology into tools that can be used in the current design process of on-board control systems, including hardware and software. This appears to be a logical next step in deploying the technology in the automotive industry in the near future.

Bibliography

[ACP91] S. Addanki, R. Cremonini, and J. S. Penberthy. Graphs of models. *Artificial Intelligence*, 51(1–3):145–177, 1991.

[And76] G. Andrews. The theory of partitions. In G.-C. Rota, editor, *Encyclopedia of Mathematics and its Applications*, volume 2. 1976.

[Bau96] H. Bauer, editor. *Automotive Handbook*. Robert Bosch GmbH, Stuttgart, Germany, 4th edition, 1996.

[BK92] D. Berleant and B. Kuipers. Qualitative-numeric simulation with Q3. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 3–16. MIT Press, Cambridge, MA, 1992.

[BKZY96] C. Bailey-Kellogg, F. Zhao, and K. Yip. Spatial aggregation: Language and applications. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI–96)*. AAAI Press, 1996.

[Bry92] R. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical Report CS–92–160, Carnegie Mellon University, School of Computer Science, 1992.

[BTC+99] P. Bidian, M. Tatar, F. Cascio, D. Theseider-Dupré, M. Sachenbacher, R. Weber, and C. Carlén. Powertrain diagnostics: A model-based approach. In *Proceedings of ERA Vehicle Electronic Systems Conference*, Coventry, UK, 1999.

[CCG+99] F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupré. Generating on-board diagnostics of dynamic automotive systems based on qualitative models. *AI Communications, Special Issue on Model-Based Reasoning*, 1999.

[CDD+00] M.-O. Cordier, P. Dague, M. Dumas, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyés. A comparative analysis of AI and control theory approaches to model-based diagnosis. In *Proceedings of ECAI'00*, Berlin, Germany, 2000.

[CN98] B. Y. Choueiry and G. Noubir. On the computation of local interchangeability in discrete constraint satisfaction problems. In *Proceed-*

ings of the 15th National Conference on Artificial Intelligence (AAAI–98), pages 326–333, Menlo Park, 1998. AAAI Press.

[Cod93] California Code of Regulations. Malfunction and diagnostic system requirements for 1994 and subsequent model-year passenger cars, light-duty trucks, and medium-duty vehicles and engines (OBD II), section 1968.1, title 13, 1993.

[Dav84] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[dJvR99] Hidde de Jong and Frank van Raalte. Comparative envisionment construction: A technique for the comparative analysis of dynamical systems. *Artificial Intelligence*, 115(2), 1999.

[dK86] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[dK92] J. de Kleer. Compiling devices: Locality in a TMS. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 295–310. MIT Press, Cambridge, MA, 1992.

[dK93] J. de Kleer. A view on qualitative physics. *Artificial Intelligence*, 59:105–114, 1993.

[dKB90] J. de Kleer and J. S. Brown. A qualitative physics based on confluences. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 88–126. Kaufmann, San Mateo, CA, 1990.

[dKMR92] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[dKW87] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[dKW89] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proceedings of the 11th IJCAI*, pages 1324–1330, Detroit, MI, 1989.

[DP88] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[DS94] O. Dressler and P. Struss. Model-based diagnosis with the default-based diagnosis engine: Effective control strategies that work in practice. In *Proc. of the 11th ECAI*, pages 677–681, Amsterdam, The Netherlands, 1994.

[DST93] J. L. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, 2nd edition, 1993.

[El 98] Y. El Fattah. An elimination algorithm for model-based diagnosis. In P. P. Nayak and B. C. Williams, editors, *Working Papers of the Ninth International Workshop on Principles of Diagnosis (DX'98)*, Cape Cod, MA, 1998.

[FF92] B. Falkenhainer and K. D. Forbus. Compositional modeling of physical systems. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 33–48. MIT Press, Cambridge, MA, 1992.

[FGN90] G. Friedrich, G. Gottlob, and W. Nejdl. Physical impossibility instead of fault models. In *Proceedings of AAAI'90*, pages 331–336, Boston, MA, 1990.

[For90] K. D. Forbus. Qualitative process theory. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 178–219. Kaufmann, San Mateo, CA, 1990.

[Fre78] E. C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 29(1):24–32, 1978.

[Fre91] E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. of AAAI–91*, pages 227–233, Anaheim, CA, 1991.

[Fre93] B. Freitag. Datenbanken 1. Lecture script (in German), Technische Universität München, Germany, 1993.

[Fre94] E. C. Freuder. Exploiting structure in constraint satisfaction problems. In B. Mayoh, E. Tyugu, and J. Penjam, editors, *Constraint Programming*, pages 51–74. Springer, Berlin, Heidelberg, 1994.

[FS92] Boi Faltings and Peter Struss, editors. *Recent Advances in Qualitative Physics*. The MIT Press, Cambridge, Massachusetts, 1992.

[FS95] E. C. Freuder and D. Sabin. Interchangeability supports abstraction and reformulation for constraint satisfaction. In *Proceedings of Symposium on Abstraction, Reformulation and Approximation (SARA'95)*, 1995.

[GLS00] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[GW89] F. Giunchiglia and T. Walsh. Abstract theorem proving. In *Proc. of the 11th IJCAI*, pages 372–377, Detroit, MI, 1989.

[GW92] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, 1992.

[Hay90] P. J. Hayes. The second naive physics manifesto. In D. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 46–63, San Mateo, CA, 1990. Morgan Kaufman Publishers.

[Hel01] Ulrich Heller. *Process-oriented Consistency-based Diagnosis — Theory, Implementation and Applications*. Ph.d. thesis, Technische Universität München, 2001.

[Hey88] John B. Heywood. *Internal Combustion Engine Fundamentals*. McGraw–Hill, 1988.

[HS97] U. Heller and P. Struss. Conceptual modeling in the environmental domain. In *Proceedings of the 15th IMACS World Congress on Scientific Computation*, pages 147–152, Berlin, Germany, 1997.

[IFS+95] Y. Iwasaki, A. Farquhar, V. Saraswat, D. Bobrow, and V. Gupta. Modeling time in hybrid systems: How fast is "instantaneous"? In *Proc. of the 14th IJCAI*, pages 1773–1780, Montreal, Canada, 1995.

[IL95] Y. Iwasaki and A. Y. Levy. Automated model selection for simulation. In *Proc. of AAAI'94*, pages 1183–1190, Seattle, WA, 1995.

[Iwa92] Y. Iwasaki. Reasoning with multiple abstraction models. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 67–82. MIT Press, Cambridge, MA, 1992.

[Kea93] Eugene L. Keating. *Applied Combustion*. Marcel Dekker Inc., New York, 1993.

[Kui86] B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, 1986.

[Kui94] Benjamin J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA, 1994.

[Kut00] A. Kutscha. Entwurf und Realisierung einer Komponente zur Abstraktion von Wertebereichen in qualitativen Modellen. Master's thesis, Department of Computer Science, Technische Universität München, 2000.

[LIM92] A. Levy, Y. Iwasaki, and H. Motoda. Relevance reasoning to guide compositional modeling. In *Workshop Notes of the 6th International Workshop on Qualitative Reasoning (QR'92)*, pages 7–21, Edinburgh, Scotland, 1992.

[Lun96] Monika Lundell. A qualitative model of physical fields. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1016–1021, Menlo Park, August 4–8 1996. AAAI Press.

[Mac92] A. K. Mackworth. The logic of constraint satisfaction. *Artificial Intelligence*, 58:3–20, 1992.

[Mau98] J. Mauss. *Analyse kompositionaler Modelle durch Serien–Parallel–Stern–Aggregation*, volume DISKI–183. Infix–Verlag, 1998.

[MS99] J. Mauss and M. Sachenbacher. Conflict-driven diagnosis using relational aggregation. In *Working Papers of the 10th International Workshop on Principles of Diagnosis (DX–99)*, Loch Awe, Scotland, 1999.

[Nay95] P. Pandurang Nayak. *Automated modeling of physical systems*, volume 1003 of *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*. Springer Verlag, New York, NY, USA, 1995.

[NJA92] P. Pandurang Nayak, Leo Joskowicz, and Sanjaya Addanki. Automated model selection using context-dependent behaviors. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 710–716, San Jose, CA, 1992. MIT Press.

[NN97] M. Nyberg and L. Nielsen. Model based diagnosis for the air intake system of the SI-engine. SAE paper 970209, Society of Automotive Engineers, 1997.

[NN98] M. Nyberg and L. Nielsen. Model based diagnosis of leaks in the air-intake system of an SI-engine. SAE paper 980514, Society of Automotive Engineers, 1998.

[Nyb99] Mattias Nyberg. *Model Based Fault Diagnosis: Methods, Theory, and Automotive Engine Applications*. Dissertation No. 591, Linköping University, 1999.

[Pla81] D. A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[Rai91] O. Raiman. Order of magnitude reasoning. *Artificial Intelligence*, 51:11–38, 1991.

[Ran98] R. Ranon. The closure properties of functional flow-based approaches and their relevance to diagnosis. In *Proc. of the 13th ECAI*, Brighton, UK, 1998.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Rei87] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[RP94] J. Rickel and B. Porter. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In *Proc. of AAAI–94*, pages 1191–1198, Seattle, WA, 1994.

[RPD90] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proc. of the 9th ECAI*, pages 550–556, Stockholm, Sweden, 1990.

[RR84] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Ellis Horwood Ser.: Math. Appl. Ellis Horwood, 1984.

[SD89] P. Struss and O. Dressler. "Physical negation": Integrating fault models into the general diagnostic engine. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI–89)*, pages 1318–1323, Detroit, MI, 1989.

[SH98] P. Struss and U. Heller. Process-oriented modeling and diagnosis — Revising and extending the theory of diagnosis from first principles. In *Working Papers of the 9th International Workshop on Principles of Diagnosis (DX'98)*, pages 110–117, Cape Cod, MA, 1998.

[SS99] P. Struss and M. Sachenbacher. Significant distinctions only: Context-dependent automated qualitative modeling. In *Working Papers of the 13th International Workshop on Qualitative Reasoning (QR'99)*, pages 203–211, Loch Awe, Scotland, 1999.

[SS00] M. Sachenbacher and P. Struss. Automated qualitative model abstraction — Theoretical foundations and practical results. In *Working Papers of the 14th International Workshop on Qualitative Reasoning (QR'00)*, Morélia, Mexico, 2000.

[SS01] M. Sachenbacher and P. Struss. AQUA: A framework for automated qualitative abstraction. In *Working Papers of the 15th International Workshop on Qualitative Reasoning (QR'01)*, San Antonio, Texas, 2001.

[SSC00a] M. Sachenbacher, P. Struss, and C. Carlén. A prototype for model-based on-board diagnosis of automotive systems. *AI Communications*, 13(2):83–97, 2000.

[SSC00b] P. Struss, M. Sachenbacher, and C. Carlén. Insights from building a prototype for model-based on-baord diagnosis of automotive systems. In *Working Papers of the 11th International Workshop on Qualitative Reasoning (DX'00)*, Morélia, Mexico, 2000.

[SSW00] 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 *Society of Automotive Engineers (SAE) World Congress*, Detroit, USA, 2000.

[Sto97] Richard Stone. *Introduction to Internal Combustion Engines*. Society of Automotive Engineers, Warrendale, PA, second edition, 1997.

[Str90] P. Struss. Problems of interval-based qualitative reasoning. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 288–305. Morgan Kaufmann, San Mateo, CA, 1990.

[Str92a] P. Struss. Diagnosis as a process. In L. Console W. Hamscher and J. de Kleer, editors, *Readings in Model-based Diagnosis.*, pages 408–418, San Mateo, CA, 1992. Morgan Kaufmann.

[Str92b] P. Struss. What's in SD? Towards a theory of modeling for diagnosis. In L. Console W. Hamscher and J. de Kleer, editors, *Readings in Model-based Diagnosis*, pages 419–450, San Mateo, CA, 1992. Morgan Kaufmann.

[Str00] P. Struss. Modellbasierte Systeme und qualitative Modellierung. In C.-R. Rollinger G. Görz and J. Schneeberger, editors, *Handbuch der Künstlichen Intelligenz*, pages 431–490. Oldenbourg Wissenschaftsverlag, München, Germany, 2000.

[Tsa93] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 1: The New Technologies*. Computer Science Press, New York, 1989.

[Wal99] D. Waltz. The importance of importance. *AI Magazine*, 20(3), 1999.

[WdK90] D. Weld and J. de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, San Mateo, CA, 1990.

[Wel88] D. S. Weld. Comparative analysis. *Artificial Intelligence*, 36:333–374, 1988.

[Wel92] D. S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56, 1992.

[WF97] R. Weigel and B. Faltings. Structuring techniques for constraint satisfaction problems. In *Proc. of the 15th IJCAI*, pages 418–423, Nagoya, Japan, 1997.

[WF99] R. Weigel and B. Faltings. Compiling constraint satisfaction problems. *Artificial Intelligence*, 115(2):257–287, 1999.

[Wil91] B. C. Williams. A theory of interactions: Unifying qualitative and quantitative algebraic reasoning. *Artificial Intelligence*, 51:39–94, 1991.

[Wil92] B. C. Williams. Interaction-based invention: Designing novel devices from first principles. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 413–433. MIT Press, Cambridge, MA, 1992.

[XML00] Extensible markup language (XML) 1.0 (Second Edition), W3C recommendation, October 2000.