AQUA: A Framework for Automated Qualitative Abstraction

Martin Sachenbacher and Peter Struss
Technische Universität München
Department of Computer Science
Orleansstr. 34, D-81667 München
E–Mail: {sachenba,struss}@in.tum.de

Abstract

In this paper, we deal with the problem of abstracting behavior models such that their level of granularity is as coarse as possible, but still fine enough to carry out a given behavior prediction or diagnosis task. The focus is on determining task–dependent distinctions within the domains of variables — i.e. qualitative values — that are both necessary and sufficient, given a model composed from a library, a granularity of possible observations, and a granularity of desired results.

We present a formalization of the problem, present fundamental results regarding the existence and characterization of solutions to task–dependent qualitative abstraction, and devise a method for automatically determining qualitative values based on a hierarchical representation of the device model that allows to exploit its specific structure.

A 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. The resulting tool set thus enhances the ability to use a behavior model of an engineered device as a common basis to support different tasks along its life cycle.

Introduction

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 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. In order to make it feasible, there are two fundamental ideas:

- to break down knowledge into model fragments such that it can be re-used in order to solve different problems in varying contexts, and
- to formulate behavior models at a level of granularity that is appropriate to provide an effective and efficient solution of the respective problem.

The former idea leads to compositional descriptions of a system's behavior (FF91). A system description (SD) consists of variables $\boldsymbol{v}$, domains $DOM(v_i)$ and relations (constraints) describing the behavior of individual components. Together, they define a relation $R(\boldsymbol{v})$ capturing the possible behaviors of the system.

The latter idea corresponds to abstraction of a behavior model. Abstraction might affect each of the constituents in SD. Thus, there are three basic types of possible abstractions: abstraction of variables, abstraction of domains, and abstraction of the relations between variables.

This paper is about automating 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. We tackle this problem in the context of domain abstraction.

Example

Consider the system depicted in figure 1. 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 $pos_{switching}$ 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 v_{pot} and v_{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 injected fuel evokes dangerous driving situations.

Assume that, for instance, v_{gnd} lies between $0V$ and $1V$, v_{batt} lies between $9V$ and $10V$, and $pos_{switching}$ equals 40%, where 0% means that the gas pedal is in rest position, and 100% means that it is fully pushed through. Consider a situation where the ECU receives

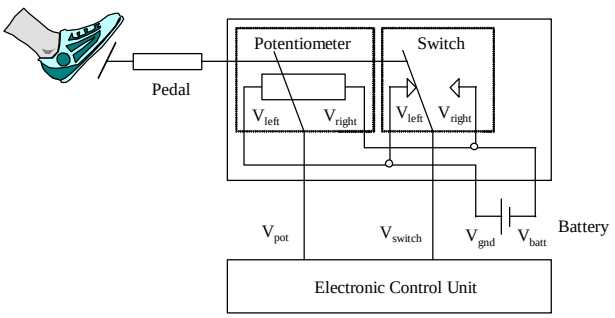

Figure 1: The Pedal Position Sensor

the following sequence of real–valued sensor readings:

$$t = 1 : v_{pot} = 2.3V, v_{switch} = 0.5V$$
$$t = 2 : v_{pot} = 2.5V, v_{switch} = 0.5V$$
$$t = 3 : v_{pot} = 3.4V, v_{switch} = 0.4V$$
$$t = 4 : v_{pot} = 5.6V, v_{switch} = 0.4V$$
$$t = 5 : v_{pot} = 6.8V, v_{switch} = 0.5V$$
$$t = 6 : v_{pot} = 8.1V, v_{switch} = 9.7V$$

The measurements reflect a delayed switch–over behavior of the switch component, as caused e.g. by a mechanical failure of this component. Suppose 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. We could use a real–valued component model for the potentiometer that precisely relates each position to a particular potentiometer voltage v_{pot}. However, this model would be overly detailed for the purpose considered. It would be unlikely to meet the tight space and run-time requirements of being part of the control unit's on-board software, for instance. Consider an abstraction of this real–valued model that abstracts voltage into three qualitative values:

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

This representation (call the corresponding system description $SD_{generic}$) is sufficient to reason about simple failures in the electrical harness, for instance, shorts to ground or battery. However, it is too coarse–grained for the above-mentioned task. At $t = t_4$ and $t = t_5$, the values for v_{pot} are subsumed by the value $[2V, 8V)$, which is consistent with the switch being either on its left or right position. Distinguishing the failure from the normal operation of the device requires an additional distinction in the domain of v_{pot}, which is not reflected in $SD_{generic}$. A more fine-grained abstraction for the domain of each voltage variable is

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

With the corresponding model SD_{base}, the failure can be discovered, because at t_5, $v_{pot} = [6V, 8V)$ is inconsistent with $v_{switch} = [0V, 2V)$. However, SD_{base} is unnecessarily fine–grained for the purpose at hand. E.g. the distinction between $[0V, 2V)$ and $[2V, 4V)$ in the domain of v_{pot} is not required for the potentiometer model. The deeper problem is that the required distinctions in the domains of the variables cannot be anticipated in generic models of the components. E.g. the separation of values for v_{pot}, as required in the example, 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. We would like to obtain component models that make just the right distinctions as required by the behavioral constraints of the device and the task the model is intended for.

Towards Qualitative Abstraction from First Principles

The example above has confronted us with the problem that simply picking model fragments from a library and composing the model is not enough. Instead, the ability to *transform* the model to the right level of abstraction *after* composing it 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 (FMEA), 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. It would be unacceptable to *manually* create models from scratch that are tailored to each of these tasks.

Deriving models that are adequate for the task at hand requires a notion of the *purpose* the model is used for, usually understood as a query on the relationship of certain variables that needs to be answered based on a model (FF91), (Nay95). However, such a notion of the purpose of a model is of limited use, regarding the aim to support different problem–solving tasks. For instance, it would not be helpful in order to construct a model 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 e.g. braking force, is above the required level. To achieve 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.

Moreover, existing approaches to automated modeling often cannot give guarantees that transformation makes the model more "optimal" with respect to the intended task, and that the transformed model covers at least the same physical behaviors as the original (base) model. For instance, in (Nay95), where the ultimate goal is to generate parsimonious causal explanations of device behavior, it cannot be guaranteed that the trans-

formed model leads to a most parsimonious causal explanation. Instead, the transformations are used as a heuristics in order to approach this goal. The devised simplifications ensure that a causal explanation can still be derived from the transformed model. Whereas these aspects are important for the kind of task pursued in this work, the resulting *approximation* not necessarily covers the same physical behaviors as the base model.

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

- the applied transformation steps must be *sound abstractions* that guarantee that the result covers at least the same physical situations as the base model (which is e.g. the case for domain abstraction);
- there has to be a notion of a *modeling goal* that allows to express what aspects of the outcome of the problem solving process we are after;
- there has to be a notion of *modeling conditions* that allows to express what inputs to the problem solving process (i.e. possible external restrictions) can occur;
- the method has to be applicable to *arbitrary relational models*, and not be limited to restricted cases such as e.g. monotonic functions.

Hence, our goal and contribution is to make explicit *task-dependency* in order to reason directly about such aspects as desired 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 qualitative abstractions of a model.

Task-dependent Qualitative Abstraction

The following sections address the problem of characterizing the adequate granularity of behavior models relative to *task-dependent* characteristics, namely

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

The set of external restrictions characterizes the available "inputs" to the model, while the set of possible solutions characterizes the "outputs" that we are interested in when solving problems using the model. Both aspects of task-dependency influence the appropriate granularity of the behavior model. They are explicitly incorporated into the model-based problem solving framework by means of two different abstractions: the identification of states that need not be distinguished, given the granularity of solutions (termed *target distinctions*), and the identification of states that cannot be distinguished, given the granularity of external restrictions (termed *observable distinctions*).

Target Distinctions

Target distinctions identify solutions that *need not* be distinguished from each other. Target distinctions 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 braking force at the wheels. 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. A target distinction corresponds to a partition:

Definition 1 (Target Distinction) *A target distinction, denoted π_{targ}, is a partition of the space $DOM(\boldsymbol{v})$, i.e. $\pi_{targ} = (\pi_{targ,1}, \pi_{targ,2}, \ldots, \pi_{targ,n})$.*

A variable v_i is said to have no target partition if the domain partition for v_i specified by π_{targ} is equal to the trivial domain partition $\pi_{triv,i} := \{DOM(v_i)\}$.

Observable Distinctions

Observable distinctions identify states that *cannot* be distinguished from each other. Similar to target distinctions, observable distinctions give rise to abstractions of a model because they introduce a "don't know" indeterminism among the behavioral states of a device. Observable distinctions reflect the granularity of external restrictions to the model. They are a means to express measurement granularity or incomplete observability of variables. The latter case occurs e.g. in on-board diagnosis, where only certain variables corresponding to the sensor inputs are observable. Note that observations are only a special case of external restrictions, which could also correspond specifications given by the user or hypothetical situations as considered e.g. in an FMEA. Analog to target distinctions, observable distinctions can be represented as a partition:

Definition 2 (Observable Distinction) *An observable distinction, denoted π_{obs}, is a partition $\pi_{obs} = (\pi_{obs,1}, \pi_{obs,2}, \ldots, \pi_{obs,n})$.*

A variable v_i is not observable at all if its domain partition $\pi_{obs,i}$ is equal to the trivial partition.

There is a duality between domain partitions and domain abstractions. A domain partition π_i can be understood as a mapping

$$\tau_i : DOM(v_i) \rightarrow 2^{DOM(v_i)}$$

from a base domain to a transformed domain that consists of sets of elements of the base domain. Depending on the situation, the first view or the second might be more convenient.

Definition 3 *A qualitative abstraction problem QAP is a tuple* $(R, \tau_{obs}, \tau_{targ})$*, where* τ_{obs} *is a domain abstraction defined by observable distinctions,* τ_{targ} *is a domain abstraction defined by target distinctions, and* R *is a relational behavior model.*

The solutions to a qualitative abstraction problem are given by a domain partition π, which corresponds to a domain abstraction τ. If τ contains sufficient distinctions to derive all information about the resulting solutions, it is denoted *distinguishing domain abstraction*:

Definition 4 *Let* $QAP = (R, \tau_{obs}, \tau_{targ})$ *be a qualitative abstraction problem. A distinguishing domain abstraction for QAP is a domain abstraction* $\tau_{ind} = (\tau_{ind,1}, \ldots, \tau_{ind,n})$ *such that for the considered external restrictions* $R_{ext} \subseteq \tau_{obs}(DOM(\boldsymbol{v}))$*, the abstracted model derives a solution* $SOL \subseteq \tau_{targ}(DOM(\boldsymbol{v}))$ *if and only if the base model derives the same solution:*

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

The requirement expressed in definition 4 means that if we are given an external restriction on the level of observable distinctions, then applying the distinguishing domain abstraction τ_{ind} *before* determining the result does not change the result on the level of the target distinction. That is, the abstracted behavior model contains *sufficient* distinctions for the task.

There might exist more than one possible domain abstraction that fulfills this criterion. In particular, the domain abstraction τ_{id} corresponding to identical mapping and the merge of the observable and target distinctions

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

contain sufficient distinctions to be distinguishing domain abstractions. Thus, we also have to state the requirement that a qualitative abstraction contains only *necessary* distinctions. A *maximal* abstraction guarantees that any finer abstraction incorporates distinctions that are unnecessary:

Definition 5 *Let* $QAP = (R, \tau_{obs}, \tau_{targ})$ *be a qualitative abstraction problem. A distinguishing domain abstraction* τ_{ind} *is a maximal distinguishing domain abstraction for QAP, if there does not exist a distinguishing domain abstraction* τ'_{ind} *for QAP such that* τ_{ind} *is a strict refinement of* τ'_{ind}*.*

A maximal abstraction incorporates only distinctions that are both necessary and sufficient according to the target and observable distinctions. It represents a level of abstraction that is 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. More precisely, it formalizes the problem of finding qualitative values for the domains of variables: finding τ_{ind} means finding sets of qualitative values for the individual variables v_i. We use the term *induced distinction* for the domain partition corresponding to τ_{ind}. An induced distinction expresses and formalizes our goal of determining qualitative values for model variables from first principles.

Definition 6 *Let* $QAP = (R, \tau_{obs}, \tau_{targ})$ *be a qualitative abstraction problem. QAP is said to be*

- *complete, if each* $R_{ext} \subseteq \tau_{obs}(DOM(\boldsymbol{v}))$ *has to be considered as possible external restriction,*
- *minimal, if* $\forall v_{i,0} \in DOM(v_i) : v_{i,0} \in \Pi_{v_i}(R)$*, i.e. none of the domains contains redundant values (see also (Tsa93));*
- *observable, if* $\forall SOL \in \tau_{targ}(R(\boldsymbol{v})), \exists R_{ext} \subseteq \tau_{obs}(DOM(\boldsymbol{v}))$ *s.th.* $SOL = \tau_{targ}(R \bowtie R_{ext})$*,*
- *consistent, if* $\tau_{obs}(R) = \tau_{obs}(DOM(\boldsymbol{v}))$*, i.e. all external restrictions are consistent with the model.*

The condition to be consistent is not a restriction, for a given QAP can be modified in such a way that this condition is satisfied. The principle is to anticipate possible revisions in the behavior model, in the extreme case by adding unknown behavior modes that correspond to unrestricted behavior of components.

Induced Distinctions vs. Interchangeability

This section shows that the problem of finding interchangeable values in constraint satisfaction (Fre91) can be reconstructed as a special case of a QAP. Fully interchangeable values define equivalence classes on the set of domain values. Thus, replacing fully interchangeable values by a single new domain value corresponds to a special case of domain abstraction:

Definition 7 *For a relation* R*, the domain abstraction* $\tau_{FI,R}$ *is defined by the domain partitions* $\pi_i = \{P_{i,1}, P_{i,2}, \ldots, P_{i,k}\}$ *given as*

$$val_1, val_2 \in P_{i,l} :\Leftrightarrow \Pi_{\ldots,v_{i-1},v_{i+1},\ldots}(\sigma_{v_i=val_1}(R)) = \Pi_{\ldots,v_{i-1},v_{i+1},\ldots}(\sigma_{v_i=val_2}(R)).$$

In $\tau_{FI,R}$, 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 w.r.t. the value for variable v_i, and are equal for all the other variables.

Proposition 1 *Let* QAP *be a complete and minimal qualitative abstraction problem such that* $\tau_{obs} = \tau_{id}$*,* $\tau_{targ} = \tau_{triv}$*. Then the maximal distinguishing domain abstraction* τ_{ind} *for QAP is equal to* $\tau_{FI,R}$*.*

Since finding fully interchangeable values is NP-hard, the proposition implies that finding induced distinctions is also NP-hard.

Determining Induced Distinctions

In the following, we assume that the given qualitative abstraction problem QAP is complete, minimal, and observable. In addition, we assume that τ_{ind} is the merge of an abstraction τ'_{obs} of τ_{obs} and an abstraction τ'_{targ} of τ_{targ}. It will be shown that in this case there exists a unique solution for QAP.

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

Theorem 1 implies that the source for abstraction of the induced distinctions can only be abstractions of the observable distinctions.

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

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

Because the observations are complete, i.e. each $OBS_k \in \tau_{obs}(DOM(\boldsymbol{v}))$ 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 elements of the set

$$\text{-}\ (R, \tau_{obs}) := \{R_{OBS,k}\}$$

form a partition of the model relation R. Let further $R_{SOL,k}$ be the solution obtained for $R_{OBS,k}$ on the level of τ_{targ}, i.e.

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

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 - (R, τ_{obs}). Let $R_{SOL,OBS,k}$ be defined by

$$R_{SOL,OBS,k} := \bigcup_j R_{OBS,j}\ \forall j \text{ s.th. } R_{SOL,j} = R_{SOL,k}.$$

Then the elements of the set of all such relations $R_{SOL,OBS,k}$, denoted

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

form a partition of the set - (R, τ_{obs}). In other words, $\Sigma(R, \tau_{obs}, \tau_{targ})$ defines a partition of the model relation that is an abstraction of the partition of the model relation defined by - (R, τ_{obs}).

Theorem 2 (Sufficient Condition) *Let QAP be a complete, minimal and observable qualitative abstraction problem. If τ_{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*

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

The following theorem presents the complete solution to the problem of deriving induced distinctions.

Theorem 3 (Complete Condition) *Let QAP be a complete, minimal, observable and consistent qualitative abstraction problem. Then τ_{ind} is given as the merge of the target distinctions with any domain partition π_i given by*

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

Theorem 2 and theorem 3 can be considered as limiting cases in a spectrum of definitions that varies on how much detailed the elements of the set $\Sigma(R, \tau_{obs}, \tau_{targ})$ are distinguished from each other. While theorem 2 considers only differences taking into account single variables, theorem 3 considers differences taking into account all 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, or the domain values lead to different solutions if combined with additional restrictions for other variables.

Computing Task-dependent Model Abstractions

The computation of τ_{ind} for a QAP involves, based on the theorems above, the subproblems of constructing the model relation R, checking (or establishing) the preconditions for applying the theorems, and computing the partition $\Sigma(R, \tau_{obs}, \tau_{targ})$.

Computation of Model Relations

Determining the model relation R can in principle be done by computing the join of the relations $R_{C,i}$ for the individual components C_i, i.e.

$$R(\boldsymbol{v}) = R_{C,1} \bowtie R_{C,2} \bowtie \ldots \bowtie R_{C,n}.$$

However, $R(\boldsymbol{v})$ can be extremely 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. Thus, we need a representation of R that is *implicit*, but still allows to efficiently carry out the necessary operations.

In engineered devices, it is typical that the interactions will be mediated through several components (i.e. defined interfaces, buses, supplies), and it is unlikely that every variable *directly* affects each other variable.

Thus, in a constraint satisfaction problem (CSP) as defined by the behavior model e.g. of an automotive system, *subproblems* can be expected to occur that are significantly smaller than the complete CSP.

Techniques referred to as solution synthesis and decomposition in constraint satisfaction (Tsa93; WF99) aim at systematically exploiting such problem–specific features. They operate on a representation of a CSP as a graph. Graph representations for CSPs can be constructed in two ways, either as a primal constraint graph or as a dual constraint graph. A dual constraint graph represents each constraint by a node (called *meta-variable*) 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 8 (Meta-Variable) *A meta–variable corresponds to a subset* $S = (v_{k1}, v_{k2} \ldots, v_{kn})$ *of variables of a ground CSP. The domain values of the meta–variable are tuples of the relation* $DOM(v_{k1}) \times DOM(v_{k2}) \times \ldots \times DOM(v_{kn})$ *with scheme* $v_{k1}, v_{k2} \ldots, v_{kn}$.

The dual constraint graph for a system description SD will be called the SD Graph. In an SD Graph, the meta–variables are given by the component behavior descriptions $R_{C,i}$, and the arcs are given by the variables in SD.

Hierarchical Clustering of System Descriptions Dechter and Pearl (DP88) observe that directional arc consistency (DAC) is sufficient for determining global consistency in a tree–structured (acyclic) CSP. Thus, the basic idea to derive the model relation from an SD Graph is to transform the SD Graph into a tree representation, even if the original SD Graph representation of the system description does not correspond to a tree. This can be achieved by systematically forming larger clusters (i.e. new meta–variables) and arranging them hierarchically in the form of a SD Tree:

Definition 9 (SD Tree) *A SD Tree is a tree whose leaf nodes are the nodes of the SD Graph, and the intermediate nodes are meta–variables corresponding to the combination of meta–variables of the SD Graph. A SD Tree is called minimal, iff each meta–variable represents only value combinations that are consistent with the model relation.*

In a SD Tree, the meta–variables can be interpreted as super–components (clusters) formed from components of the original system description. Clustering is accomplished by repeatedly identifying cliques in the SD 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. A special case occurs when the considered clique consists of a single node, i.e. when the arcs to be eliminated are self–arcs that refer to a single meta–variable. In this case, the corresponding node in the SD Tree (which would otherwise have a branch factor of one) can be represented together with its child node as a single entity by attributing nodes x in a SD Tree both with a meta–variable $MV(x)$ as outlined above and a meta–variable $MV_{shared}(x)$ with reduced scheme comprising only variables further shared with other meta–variables:

Procedure Reduce(x,ns)

$$s := scheme(MV_{shared}(x)) = scheme(MV(x)) \backslash ns,$$
$$DOM(MV_{shared}(x)) = \Pi_s(DOM(MV(x))).$$

Algorithm (Generation of SD Tree)

Step 1 For each node x in the SD Graph, let ns be labels of self–arcs of x. Call $Reduce(x, ns)$. Remove self–arcs of x in the SD Graph.

Step 2 Identify a clique in the SD Graph.

Step 3 Build a new node y for clique $x_1, x_2, \ldots, x_k$ connected by arcs labeled with variables s. Set

$$scheme(MV(x)) = \bigcup_{i=1,\ldots,k} scheme(MV(x_i)),$$
$$DOM(MV(x)) = \bowtie_{i=1,\ldots,k} DOM(MV_{shared}(x_i)).$$

The nodes $x_1, x_2, \ldots, x_k$ become the children of y in the SD Tree.

Step 4 Replace the clique $x_1, x_2, \ldots, x_k$ by y in the SD Graph.

Step 5 Call procedure $Reduce(y, s)$.

Step 6 Proceed with step 1 until there are no more arcs left in the SD Graph.

In a SD Tree, for each variable v_i in the system description, there exists exactly one node x such that $v_i \in scheme(MV(x)) \setminus scheme(MV_{shared}(x))$. SD Tree generation can be viewed of 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 SD Tree is bound by $c + s - 1$, where c is the number of components and $s \cdot \ n$ the number of shared variables in the system description. For a given system description, different SD Trees are possible, corresponding to different ways of choosing cliques in step 2 of the algorithm.

Minimality of the SD Tree is achieved by establishing directional arc consistency between the meta–variables of the SD Tree. The complexity for this operation is $O(m \cdot k^2)$, where m is the number of meta–variables in the tree, and k the (maximum) domain size of meta–variables. A minimal SD Tree implicitly represents the model relation R. Figure 2 shows an example of a SD Tree for the pedal position sensor (the ECU component is not contained in the model).

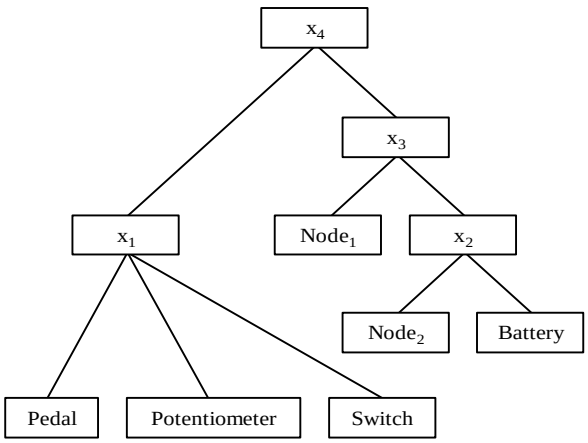

Figure 2: SD Tree for the Pedal Position Sensor

Computing Induced Distinctions

Basic operations necessary to compute - (R, τ_{obs}) and $\Sigma(R, \tau_{obs}, \tau_{targ})$ can be carried out efficiently using an SD Tree, without the need to refer to the base model relation R explicitly. Computing - (R, τ_{obs}) amounts to determining the partition elements $R_{OBS,k} \subseteq R$ that are consistent with different tuples of $\tau_{obs}(DOM(\boldsymbol{v}))$. Computing $\Sigma(R, \tau_{obs}, \tau_{targ})$ amounts to determining, in turn, partition elements $R_{SOL,OBS,k}$ for - (R, τ_{obs}) that are consistent with different tuples of $\tau_{targ}(DOM(\boldsymbol{v}))$. For a partition $\pi = (\pi_1, \ldots, \pi_n)$ corresponding to a domain abstraction τ, the following algorithm determines the partition elements of R that are consistent with different tuples of $\tau(DOM(\boldsymbol{v}))$:

Algorithm (SD Tree Partition)

Step 1 For each node x, set the partition π_x of $DOM(MV(x))$ equal to the trivial partition.

Step 2 Choose a variable v_i and a partition element $P_i \in \pi_i$.

Step 3 Determine $\sigma_{v_i=P_i}(R)$. For each meta–variable $MV(x)$, this yields a partition $\pi'_x = \{P_{con}, P_{incon}\}$ where P_{con} denotes the domain values consistent and P_{incon} the domain values inconsistent with P_i.

Step 4 Merge π_x with π'_x.

Step 5 Proceed with step 2 until all variables and all partition elements of π have been considered.

Using the algorithm above, - (R, τ_{obs}) is obtained as the SD Tree partition for π_{obs}. $\Sigma(R, \tau_{obs}, \tau_{targ})$ can in turn be obtained as the SD Tree partition for π_{targ} "on top" of the SD Tree partition - (R, τ_{obs}).

The implicit representation of - (R, τ_{obs}) and $\Sigma(R, \tau_{obs}, \tau_{targ})$ through partitions of meta–variables in the SD Tree collapses the overall number of partition elements that are needed to be represented. While e.g. - (R, τ_{obs}) can contain as many elements as tuples in R, there are maximally $m \cdot k$ partition elements in an SD Tree, where is m the number of nodes in the SD Tree and k the maximal domain size of a meta–variable. The result is an implicit representation of the partition $\Sigma(R, \tau_{obs}, \tau_{targ})$ through partitions π_Σ of the meta–variables in the SD Tree.

Theorem 4 *For a model relation R given as an SD Tree, $\Sigma(R, \tau_{obs}, \tau_{targ})$ can be computed in time polynomial in n, o, t, m and k, where n is the number of variables, o and t the maximal number of elements in the domain partitions $\pi_{obs,i}$ and $\pi_{targ,i}$, m the number of meta–variables in the SD Tree, and k the maximal domain size of a meta–variable.*

The necessary preconditions for applying the theorems in the previous section (definition 6) to a qualitative abstraction problem QAP can be checked with the same effort (see (Sac01)). For instance, minimality can be determined (or, alternatively, be established) by projecting each meta–variable on the variables occuring in its scheme.

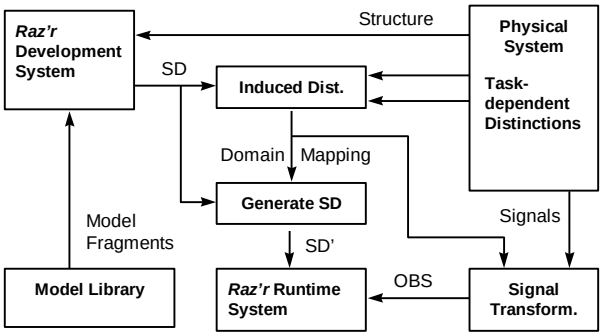

Figure 3: Components of the AQUA framework

A Prototypic System for Task-dependent Qualitative Abstraction

The methods outlined in the previous sections have been implemented in a prototypic system termed AQUA (Automated Qualitative Abstraction). It builds on components of an existing model–based reasoning framework called Raz'r that consists of a development system for defining domains, constraint types and device structures for composing system descriptions and a runtime system for performing behavior prediction and diagnosis. In addition to the basic Raz'r components, AQUA consists of (figure 3)

- a component for *Computation of Induced Distinctions* that determines task-dependent qualitative abstractions based on the algorithms described above;
- a *System Description Generator* that applies domain abstractions to a (real–valued or finite) system description, transforming it (in particular, the involved constraint types) to the specified level of granularity;
- a *Signal Transformation Module* that generates qualitative observations by applying domain abstractions to numerical (time–varying) data.

Using AQUA, several tasks can be supported in the context of model–based problem solving that, up to now, essentially had to be carried out manually. Consider again the device shown in figure 1. For the base model SD_{base}, the domain was

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

for variables involving voltage. Assume

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

to be the domain for variables involving position. The fact that the control unit observes the signal from the potentiometer and the signal from the switch can be expressed by the observable distinction

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

The target distinctions are determined by the goal to distinguish between the domain values for the variable v_{switch} (assume the plausibility check itself is not represented in the model):

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

Based on these inputs, AQUA determines a partition for v_{pot} that consists of three partition elements:

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

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 $5.6 \cdot 10^7$. Compared to the model $SD_{generic}$, 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. For the granularity of the base model $SD_{transform}$, the signal transformation component yields four qualitative observations at time points $t = 1$, 4, 5 and 6, respectively. For time point $t = 5$, the runtime system detects an inconsistency of $SD_{tranform}$ with the observations. The runtime for diagnosis is 0.256 seconds, which is about 43% less than for the finer–grained model SD_{base}, which corresponds roughly to the reduction of the domain size of v_{pot}. This example illustrates AQUA's ability to support the modeling problem of determining distinctions for the domains of variables that are essential for a certain task.

Discussion

This paper identifies fundamental properties of qualitative abstraction that is based on domain abstraction and reveals relationships between qualitative reasoning and constraint satisfaction techniques to structure problems and compactly describe their solutions (Fre91), (WF99).

AQUA offers a principled way to turn base behavior models (also real–valued models, as common in industrial applications) into qualitative models in order to make them amenable to model-based problem solving 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 as one of the major roadblocks to a more wide–spread use of model–based reasoning techniques.

QSIM and its extensions (Kui94) incorporate methods for deriving landmarks and performing semi-quantitative reasoning based on a behavior model, however, only within the context of *simulation* of device behavior over time. Another difference is that AQUA can process arbitrary relations and is not limited to pre–defined algebraic constraints such as addition or multiplication. E.g., (Sac01) presents an application where qualitative values are derived for a model involving a diesel engine described by a characteristic map.

The methods devised for computing qualitative abstractions are based on the SD Tree as a data structure that "compiles" a model relation into a an implicit representation in order to avoid combinatorial explosion.

As the computational complexity of deriving induced distinctions with an SD Tree can be bound to structural properties of the system description, this offers a basis to identify tractable subsets of qualitative abstraction problems. The principle is that if the maximal size of meta–variables is bounded, then the complexity of deriving task–dependent qualitative abstraction is polynomial (see also theorem 4). Resistive networks (Ran98) are a special class of devices that can be modeled by component types describing the generation, transportation and consumption of energy. For clustering such component types, the resulting super–components have the same relation types as the original component relations (called "closure property" in (Ran98)). Consequently, for behavior models describing resistive networks, the maximal size of meta–variables is bounded by the size of the original component relations, and hence such devices correspond to a class of problems for which task–dependent qualitative abstraction is tractable.

Acknowledgments

This work benefited from discussions with Oskar Dressler, Daniele Theseider-Dupré, and Johan de Kleer. It has been supported by the European Union through contract No. BE 95/2128 and by the German Ministry of Education and Research through contract No. 01 IN 50941.

References

Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

B. Falkenhainer and K. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. of AAAI-91*, pages 227–233, Anaheim, CA, 1991.

B. J. Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA, 1994.

P. Pandurang Nayak. *Automated modeling of physical systems*. Lecture Notes in Computer Science. Springer Verlag, New York, NY, USA, 1995.

R. Ranon. The closure properties of functional flow-based approaches and their relevance to diagnosis. In *Proc. of the 13th ECAI*, Brighton, UK, 1998.

M. Sachenbacher. *Automated Qualitative Abstraction and its Application to Automotive Systems*. Ph.D. thesis, Technische Universität München, 2001.

E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

R. Weigel and B. Faltings. Compiling constraint satisfaction problems. *Artificial Intelligence*, 115(2):257–287, 1999.