Design and Realization of a Knowledge-based System that Supports the Setting into Operation of Hydraulic Systems

H. Kleine Büning, Benno Stein

tr-ri-95-166

Design and Realization of a Knowledge-based System that Supports the Setting into Operation of Hydraulic Systems

Technical Report
June, 1995

H. Kleine Büning   Benno Stein

This work was supported by the DFG

Project number: Kl 529/3-1 and Kl 529/3-2

University of Paderborn
Department of Mathematics / Computer Science
33095 Paderborn, Germany
email: kbcsl@uni-paderborn.de

Report tr-ri-95-166
Series Computer Science
August, 1995

Abstract

The task of setting a hydraulic system into operation constitutes some kind of configuration problem. To be more specific, this task is comprised of several *configuration checking problems* placed at different levels of abstraction.

Configuration technology as developed in the past is only partly capable to cope with these checking problems. The reason for this is that the checking of a hydraulic system encloses the composition and simulation of *behavior* descriptions. By contrast does standard configuration technology usually deal either with objects that are described at a more abstract level or with systems whose composition process can be structured in a particular way.

This report describes how the checking of hydraulic systems can be automated. First, it develops a formal basis for behavior-based configuration problems, from which the hydraulic checking problem represents an instance. Grounding in this formal framework the necessary concepts for the representation and processing of hydraulic knowledge are introduced. A large part of these concepts has been operationalized in the knowledge-based system $\overset{\text{art}}{\textit{deco}}$, which will be sketched out as well.

The last chapter of this report addresses the diagnosis of hydraulic systems.

Contents

Chapter 1

Introduction

Overview of the Report

The task of setting a hydraulic system into operation constitutes some kind of configuration problem. To be more specific, this task is comprised of several *configuration checking problems*, placed at different levels of abstraction [25], [26].

Configuration technology as developed in the past is only partly capable to cope with these checking problems. The reason for this is that the checking of a hydraulic system encloses the composition and simulation of *behavior* descriptions. By contrast does standard configuration technology usually deal either with objects that are described at a more abstract level or with systems whose composition process can be structured in a particular way.

Thus, in first place, we need a formal framework that precisely defines what "configuration based on behavior" means. Such a framework is presented in chapter 2. In chapter 3 we then introduce the checking of hydraulic systems as an instance of the generic behavior-based checking problem, as well as developing the basic principles for an automation of the hydraulic checking procedure. A large part of our concepts has been operationalized in the knowledge-based system $\overset{\text{art}}{\textit{deco}}$. Chapter 4 outlines the philosophy and some inference concepts of this system.

Of course, checking the function of a hydraulic system will not result in a simple short answer, "OK" or "not OK". Instead, different classes of faults may be detected and indicated, such as an incorrect switching logic or exceeded value ranges. In addition to diagnosis information of that simple type, which can be generated when solving the different checking problems, we investigated the diagnosis of hydraulic systems in more generally terms; a diagnosis system can provide for additional support when setting a hydraulic system into operation. Chapter 5 contributes to this problem field.

The remainder of this chapter shortly reviews the notions of configuration, configuration systems, and knowledge-based systems; configuration tasks are usually problems knowledge-based systems deal with.

Configuration and Configuration Systems

Common definitions for configuration describe it as the process of composing a technical system from a predefined set of objects. The result of such a process is called configuration too and has to fulfill a set of constraints given. Aside from technical restrictions, a customer's demands constitute a large part of these constraints [38], [46].

This informal definition gives a declarative description of the job a configuration system does; it motivates the configuration process from its result. Figure 1.1 depicts this view: $Q_0, \dots, Q_e$ denote different states of a system being configured, where Q_0, the starting point, is the empty set, while Q_i comprises the *qualities* of the system after configuration step i. Q_e denotes the qualities of the readily configured system. The transformation steps t_j stand for the operations performed by the configuration system within step j.

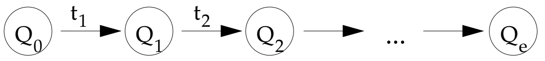


Figure 1.1: *Configuration as a sequence of states and transitions*

A configuration system can be seen as a program that takes a set of demands $D \subseteq Q_e$ as input and computes all information to describe *extensionally* the configured system. I.e., it generates information about the required objects, their type, number, topology etc. such that the emerging configuration fulfills all constraints (cf. figure 1.2).

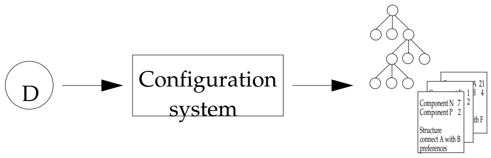


Figure 1.2: *The job a configuration system performs*

The definition above does not imply information about the complexity of configuration problems or how to get the hang of them. This should not be surprising, the development of a configuration system requires the analysis of the domain and the processing of related problems, more or less. As a consequence, we cannot provide a set of "general purpose configuration algorithms" nor build a generic configuration platform.

Research in the field of configuration can be divided in the following manner: approaches that focus on an abstract process of configuration and approaches that aim at adequate mechanisms to specify configuration knowledge. Clearly, when tackling a real-world problem, none of these approaches could do a complete job. In the former the knowledge has to be integrated—just as with the latter it has to be processed. Employing domain knowledge within a configuration system can affect both fields of research.

Knowledge-based Systems

In order to tackle a configuration problem, expertise needs to be operationalized. Thus, most of the configuration systems developed in the past were called *knowledge-based* configuration systems or *expert systems* of configuration. This subsection contains a brief introduction to knowledge-based systems. This information may be useful to clear up the scepticism that sometimes is associated with knowledge-based systems or related terms.

Highly sophisticated and complex computational methods can be found especially in engineering domains. However, a set of differential equations describing a physical system along with the numerical methods solving them should not be called a "knowledge-based system" or "expert system". Also the programming techniques that were used to realize a system, e.g. rule-based or frame-based techniques, are not a useful criterion as to whether a program is an expert system.

Brown and Chandrasekaran define expert systems as programs where we can find some kind of computations that underly intelligent behavior and which therefore are discrete, symbolic, and qualitative [2]. They call this kind of computation *problem space exploratory techniques* and characterize them to be "intelligent" in the following sense:

> *"They explore a problem space, implicitly defined by a problem representation, using general search strategies which exploit typically qualitative heuristic knowledge about the problem domain."*
>
> Brown & Chandrasekaran, [2], p.4

In fact such computational methods make up an important part of expertise. In a nutshell, we will refer to a program as a knowledge-based system if it operationalizes the knowledge, the experiences, or the procedures of an expert in whole or in part.

The classical view of knowledge-based systems is the following:

$$\begin{aligned} \text{knowledge-based system} &= \text{domain-independent inference engine} \\ &+ \text{domain-specific knowledge base} \\ &+ \text{problem-specific database} \end{aligned}$$

I.e., an important idea of knowledge-based systems is the distinction between domain knowledge on the one hand and the methods that process this knowledge on the other: Inference mechanisms are applied to the knowledge in the knowledge base and are intended to produce a solution of the actual problem. Knowledge base and inference component may be completed by modules that guide the user, generate explanations, or realize the specification of new knowledge.

A point of criticism related to this view is the explicit separation of knowledge base and inference mechanisms. Actually, a large part of an expert's knowledge needs tailored algorithms determining *how* the knowledge is to be processed. Bylander and Chandrasekaran coined the term *interaction hypothesis* in this context [3].

Thus, a knowledge base will not contain merely facts and rules but also algorithms that realize both the processing and the specification of expertise. This understanding leads to a view of knowledge-based systems as shown in figure 1.3.

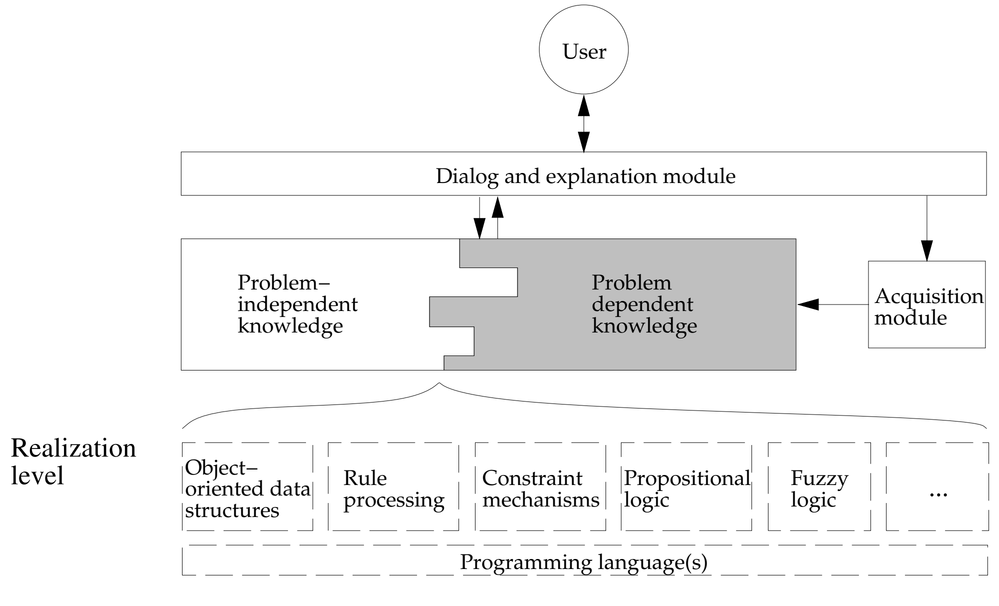


Figure 1.3: *Alternative view of a knowledge-based system*

Note that the knowledge base is divided into a problem-*in*dependent and a problem-dependent part. The former part models the never changing concepts and theories of a domain—example: the descriptions and the algorithms that operationalize Kirchhoff's or Ohm's law. The other part can be filled with knowledge less fundamental for the domain and the problem respectively, i.e., it stores definite situations or new dependencies. The boxes below the knowledge base designate some exemplary techniques that *can* be used to operationalize knowledge. In particular, there is no explicit inference module apart from the knowledge base.

In order to develop such a system, the mechanisms for knowledge specification and knowledge processing have to be oriented by the actual problem. Thus, from today's point of view, it is hardly possible nor very useful to develop universal, that is, domain-independent problem solving systems.

Chapter 2

Formal Framework

Behavior-based Configuration

The former section introduced configuration as some kind of selection problem: Given is a set of objects where the task is to select objects such that the required demands are fulfilled. Such a view of configuration can be misleading since it neglects additional jobs that may come up when synthesizing a system.

Suppose e.g., we had to configure a system where the set D of demands contains complicated behavior prescriptions. Selecting and putting together objects such that the resulting system provides the desired behavior requires (*i*) the processing of behavior descriptions that base on physical connections, and (*ii*) experience and creativity to control the process of selecting and connecting[1].

In this place we give a formal definition of such behavior-based configuration problems. Notice that presently there exists no general theory of how the creativity that guides a process of configuration or design can be operationalized. Thus, later within this section, we will define the less complex *checking problem*.

Definition 2.1 (Configuration Problem Π***).*** A configuration problem under a behavior-based model (Π) is a tuple $\langle O, F, V, B, T, D\rangle$ whose elements are defined as follows:

- O is an arbitrary, finite set of objects. It is called the *object set* of Π.
- F is an arbitrary, finite set of functionalities. It is called the *functionality set* of Π. Each element in F denotes a (possibly constant) *function* in the parameter "time". For the sake of simplification, we will normally refer to $f(t) \in F$ as f.
- For each functionality $f \in F$ there is an arbitrary, possibly infinite set v_f, called the *value set* of f. The elements of v_f are partial functions in the parameter time; i.e., they are defined on a subset of $\mathbf{R}^+$. $V = \{v_f \mid f \in F\}$ is comprised of these value sets.

[1] Within engineering domains, the process described under (*ii*) is called *model formulation*.

- For each object $o \in O$ there is an arbitrary finite set B_o, called the set of *behavior constraints*. Each behavior constraint $b \in B_o$ defines a relation on $v_{f_{b_1}} \times v_{f_{b_2}} \times \ldots \times v_{f_{b_k}}$ where $f_{b_i} \in F$, $\{b_1, b_2, \ldots, b_k\} \subseteq \{1, 2, \ldots, n\}$, and $n = |F|$. Such a relation may be specified by a function or by a possibly infinite set of tuples. $B = \{B_o \mid o \in O\}$ is comprised of the sets of behavior constraints.

  Based on the definition of behavior constraints, we agree on following notions:

  (*i*) $F_b = (f_{b_1}, f_{b_2}, \ldots, f_{b_k})$ is the tuple of the functionalities associated to the value sets $v_{f_{b_i}}$ where b is defined upon.

  (*ii*) A tuple of functions $X = (x_1, x_2, \ldots, x_k)$, $x_i \in v_{f_{b_i}}$, $f_{b_i} \in F_b$ *matches (fulfills)* the behavior constraint b, if X stands in the relation defined by b.

- For each functionality f there is a *test* t_f, which is a partial function $t_f : v_f \times v_f \to \{\mathit{True}, \mathit{False}\}$. A test t_f specifies under what condition a demand (see below) is fulfilled. $T = \{t_f \mid f \in F\}$ is comprised of all tests.

- D is an arbitrary, finite set of demands. Each demand $d \in D$ is a pair (f, x) where $f \in F$ and $x \in v_f$. If x is an invariant function of time, the demand will be called "stationary"; otherwise, the demand will be called "dynamic".

Remarks. For example, if we wanted to design an electrical circuit, typical objects in O would be resistors, capacitors, etc. The functionalities in F would specify the typical characteristics of the objects such as electrical resistances or capacities. Behavior constraints in this example would be Ohm's law and other electrotechnical regularities defined upon the functionalities in F.

Definition 2.2 (Configuration). Let $\Pi = \langle O, F, V, B, T, D\rangle$ be a configuration problem. A configuration is a triple $C = \langle E, Q, S\rangle$ where $E \subseteq O$ is a set of objects, Q is a quality set with tuples (f, x) where $f \in F$ and $x \in v_f$, and S is a set of tuples (f, g) with $f, g \in F$. E specifies those *elements*, where the configured system is to be realized with. A quality $(f, x) \in Q$ assigns the possibly constant function x to the functionality f. S defines the *structure* of C by means of the functionality-pairs to be unified. Additionally, we claim the following conditions to hold:

(*i*) Let $F_E = \{f \mid f \in F_b,\ b \in B_o,\ o \in E\}$ comprise all functionalities of C. Then, Q must be both *definite* and *complete* with respect to F_E, i.e.: For each functionality $f \in F_E$, there exists exactly one functionality-value-pair $(f, x) \in Q$.

(*ii*) If $(f, x), (g, y) \in Q$ and $(f, g) \in S$ then $x = y$.

(*iii*) Let $B_E = \{b \mid b\ \in B_o,\ o \in E\}$ comprise all behavior constraints of C. Then, Q must be *correct* with respect to B_E, i.e., to each $b \in B_E$ must apply:
The tuple of functions $X = (x_1, x_2, \ldots, x_k)$, induced by the functionality-value-pairs $(f_{b_i}, x_i) \in Q$, $f_{b_i} \in F_b$, matches (fulfills) the behavior constraint b. I.e., X stands in the relation defined by b.

Remarks. When solving a configuration problem Π under a behavior-based model, aside from a selection problem, also a structure definition problem, a model formulation problem, and a model processing problem have to be solved. It has to be determined which functionalities of the components selected have to be unified in order to achieve a behavior that fulfills all demands. Such a unification is equivalent to a connection of the objects in a physical sense and is specified by S: Each element $(f, g) \in S$ indicates that the functionality f is to be unified with the functionality g, i.e., $f \equiv g$.

The conditions (*i*) – (*iii*) guarantee the technical correctness of a configuration C, that is to say, if C can be realized from its topology and its behavior: Condition (*i*) establishes that there is a definite specification for all functionalities of C. Condition (*ii*) claims each two functionalities being equal if they are unified. Condition (*iii*) guarantees that all behavior constraints of C can be fulfilled by the functionality-value-pairs in Q.

Note that the functional dependencies within Π are represented by local constraints and a list of functionalities to be unified. These constraints and the information about unification form the input for a model synthesis process[2] that in turn yields a mathematical description of the system. In order to compute Q, which describes the system's global behavior, this mathematical description has to be processed by direct or by iteration methods.

Definition 2.3 (Solution of** Π**). A configuration $C = \langle E, Q, S\rangle$ is a solution of a configuration problem $\Pi = \langle O, F, V, B, T, D\rangle$ if and only if the following conditions hold:

(*i*) C is a configuration according to *Definition 2.2*, i.e., C is technically correct.

(*ii*) For each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \mathit{True}$.

Remarks. A solution of a configuration problem Π specifies which objects have to be selected, how they are parameterized, and how they are connected. Condition (*i*) claims a configuration's correctness while condition (*ii*) guarantees all demands being fulfilled according to the associated test predicates in T. Usually exists more than one solution of a configuration problem Π.

Example

The following example establishes an instance of the behavior-based configuration problem Π. Let us consider we had to design a simple electrical circuit. Given are resistors, capacitors, and inductive coils. The goal is to *select*, *connect*, and *parameterize* the components in such a way that we obtain a damped oscillating circuit with a frequency of about $10\mathit{kHz}$ and a time constant of $0.2s$. Figure 2.1 illustrates the task.

In order to keep this example clear, we refrain from the specification of wires. Even such a—from the electrotechnical standpoint—simple problem needs more than a complete enumeration of possible subsets of the components: The design process is guided by the

[2] We will take up this term in chapter 3.

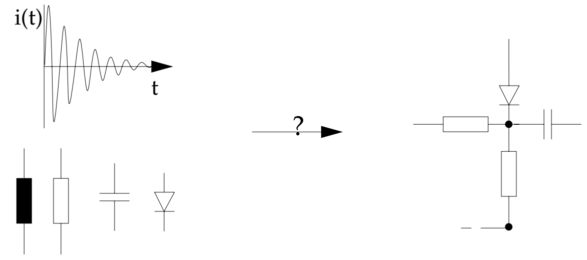


Figure 2.1: *Designing a simple circuit*

experience and the knowledge about physical connections. A formal specification of Π related to our example is the following:

1. $O = \{$R (= *resistor*), C (= *capacitor*), L (= *coil*)$\}$
2. $F = \{$ resistance, $\mathrm{U1_R}$, $\mathrm{U2_R}$, $\mathrm{i1_R}$, $\mathrm{i2_R}$, capacity, $\mathrm{U1_C}$, $\mathrm{U2_C}$, $\mathrm{i1_C}$, $\mathrm{i2_C}$, $\mathrm{Q_C}$, inductivity, $\mathrm{U1_L}$, $\mathrm{U2_L}$, $\mathrm{i1_L}$, $\mathrm{i2_L}\}$

   Each element $f \in F$ is a real function; $f : \mathbf{R}^+ \to \mathbf{R}$.
3. $v_f = \mathcal{C}(\mathbf{R})$, the set of continuous functions on $\mathbf{R}$, $f \in F,\ V = \{v_f \mid f \in F\}$
4. $B_\mathrm{R} = \{$R-voltage, R-current$\}$
   $:= \{\mathrm{U1_R} - \mathrm{U2_R} = \text{resistance}\cdot\ \mathrm{i1_R},\ \mathrm{i1_R} = \mathrm{i2_R}\}$

   $B_\mathrm{C} = \{$C-voltage, C-current, C-charge$\}$
   $:= \{\mathrm{U1_C} - \mathrm{U2_C} = \text{capacity}\cdot\ \mathrm{Q_C},\ \mathrm{i1_C} = \mathrm{i2_C},\ \frac{d\mathrm{Q_C}}{dt} = \mathrm{i1_C}\}$

   $B_\mathrm{L} = \{$L-voltage, L-current$\}$
   $:= \{\mathrm{U1_L} - \mathrm{U2_L} = \text{inductivity}\cdot\ \frac{d\mathrm{i1_L}}{dt},\ \mathrm{i1_L} = \mathrm{i2_L}\}$

   $B = \{B_\mathrm{R},\ B_\mathrm{C},\ B_\mathrm{L}\}$
5. $t_f(\chi, \gamma) :\ \|\chi - \gamma\| \leq 0.1,$ with $f \in F,$ and $\chi, \gamma \in v_f,$
   $T = \{t_f \mid f \in F\}$
6. $D = \{(\mathrm{i1_R},\ e^{-5t} \cdot sin(2\pi \cdot 10^4 \cdot t))\}$. This definition determines the circuit's oscillating frequency and its time constant.

A possible solution of the configuration problem is given in figure 2.2. It determines a function for each parameter and fulfills both all constraints and all demands.

Remarks. The solution of such a problem can hardly be found by a generate-and-test procedure but needs knowledge about model formulation in electrical engineering. To solve the above problem, a human expert would perform the following steps:

1. Identification of the circuit's topology.

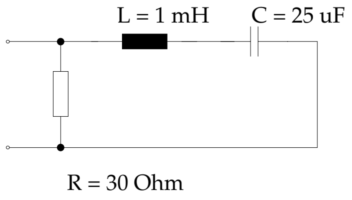


Figure 2.2: *A solution of the example*

2. Development of a a global behavior model from the local behavior constraints. The subsequent differential equation represents a correct model regarding our example: $\mathrm{L}\cdot \ddot{\mathrm{i}} + \mathrm{R}\cdot \dot{\mathrm{i}} + \frac{\mathrm{i}}{\mathrm{C}} = 0$
3. Solution of the differential equation and evaluation of the resulting terms for the frequency and the time constant.

Checking a Configuration's Behavior

As mentioned above, the creative design process can be automated to a small part only. But, the complexity of a behavior-based configuration problem will be definitely reduced, if we restrict ourselves to the *checking* of a given configuration. Often, even such a checking problem turns out to be very sophisticated since we have to automate a model formulation process that founds on physical relations and to process the resulting model. Chapter 3 introduces such a behavior-based checking problem and describes how it can be solved. In the following, we define the checking problem precisely.

Definition 2.4 (Checking Problem Π^c***).*** A checking problem under model M3 (Π^c) is a tuple $\langle O, F, V, B, T, D, S\rangle$ whose elements are defined as in *Definition 2.1* and in *Definition 2.3* respectively.

Remarks. A checking problem Π^c defines a set of objects, their local behavior concepts, and how the objects are connected. I.e., a specification of a technical system and a set of demands are given where the goal is to *check* whether the system fulfills these demands or not.

Definition 2.5 (Solution of Π^c***).*** A solution of a checking problem $\Pi^c = \langle O, F, V, B, T, D, S\rangle$ is a set Q containing tuples (f, x) where $f \in F$ and $x \in v_f$. Q is called a solution of Π^c if and only if the following conditions hold:

(*i*) $C = \langle O, Q, S\rangle$ is a configuration according to *Definition 2.2*, i.e., C is technically correct.

(*ii*) For each demand $d = (f, x) \in D$ there exists a quality $q = (g, y) \in Q$ such that $f = g$ and $t_f(x, y) = \ \mathit{True}.$

Remarks. This definition is similar to *Definition 2.3*. As a difference to the above, all objects of O are used to compose the system to be checked. Π^c can be viewed as some kind of *constraint satisfaction problem*: The sets O and S establish a network of nodes where each node is characterized by a functionality $f \in F$. B and D define the constraints of this network and may be of both numerical or symbolic type.

Each set Q that is a solution of Π^c defines a function $\gamma : F \rightarrow \bigcup v,\ v \in V$:

$$Q = \{(f_1, \gamma(f_1)),\ (f_2, \gamma(f_2)), \ldots, (f_n, \gamma(f_n))\}$$

In other words, solving this constraint satisfaction problem means to determine γ. If no such function exists, the checking problem Π^c will be contradictory, i.e., the specified system does not fulfill the desired demands. A checking problem will be underspecified, if more than one function γ exists.

Chapter 3

Checking of Hydraulic Systems

The configuration of hydraulic systems founds on deep physical connections, it needs creativity, and, at the present time, it cannot be automated.

The *checking* of hydraulic systems founds on the same physical connections; but although it doesn't need creativity, it cannot be automated straightforward. This chapter presents a new view of the configuration of hydraulic systems; it illustrates those parts of the configuration process which make up the checking problem, and it shows how to get a grip on them.

Section 3.1 provides for introductory information. It becomes clear that an *automated analysis* of hydraulic systems will be the major engine when checking hydraulic systems. Thus, we investigate in section 3.2 the analysis step in more detail. Section 3.3 presents a generic component model that allows the formulation of hydraulic checking and parameterization problems, and section 3.4 addresses the processing of this component model. Here we show how the analysis step and, as a consequence, the checking and parameterization problem in hydraulics can be automated. In addition to the inference concepts in section 3.4, we develop in section 3.5 a preprocessing approach for a particular class of hydraulic behavior constraints.

3.1 Placement of the Problem

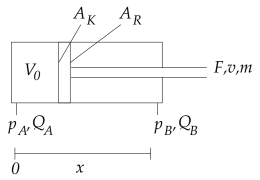
Hydraulic systems are used to perform various kinds of manipulation tasks. Such a task can be a lifting problem, the actuation of a press, or the realization of a robot's kinematics. A hydraulic system consists of hydraulic and, eventually, some mechanical and electronic components.

Hydraulic Components

Hydraulic components are the building blocks of the hydraulic checking problem. So it is useful to introduce some of their underlying physical principles to convey an idea of the configuration process's complexity. Note that hydraulic engineering is a domain which cannot be treated in detail here. We refer to [27] and [25] where deeper information relating hydraulics is provided.

Hydraulic components can be divided into three classes: (*i*) work components like cylinders, (*ii*) control components like valves, and (*iii*) service components such as pumps, tanks, and pipes. All components are described by their stationary and their dynamic behavior.

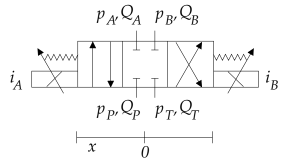
Cylinders are the actuators of a hydraulic system; they transform hydraulic energy into mechanical energy. Valves in the form of relief valves, throttle valves, proportional valves, or directional valves control flow and pressure of the hydraulic medium. Pumps provide the hydraulic energy, i.e., the necessary pressure p and flow Q. Figure 3.1 and 3.2 show the basic structure of a differential cylinder and a proportional valve respectively. Below these figures a small extract of the related behavior description is given.



A_K, A_R	= Piston areas	p	= Pressure
d_z	= Piston friction	Q	= Flow
E'_{Oil}	= Oil elasticity	V_0	= Cylinder volume
F	= Force	v	= Velocity
m	= Mass	x	= Elongation

Figure 3.1: *Basic structure of a differential cylinder*

$$\begin{array}{lcll}
F & = & p_A \cdot A_K - p_B \cdot A_R - d_z \cdot v & \text{(stationary force balance)} \\
Q_A & = & A_K \cdot v & \text{(continuity condition)} \\
F(t) & = & p_A(t) \cdot A_k - p_B(t) \cdot A_R - m \cdot \dot{v}(t) - d_z \cdot v(t) & \text{(force balance)} \\
\dot{p}_A(t) & = & \frac{E'_{Oil}}{V_0 + A_K \cdot x(t)} \cdot (Q_A(t) - A_K \cdot v(t)) & \text{(pressure rise)}
\end{array}$$



i	= Current		
K_p	= Pressure gain		
p	= Pressure		
Q	= Flow		
x	= Elongation		

Figure 3.2: *Basic structure of a proportional valve*

$$\begin{array}{lcll}
x & = & K_p \cdot (i_A - i_B) & \text{(position of valve piston)} \\
R_h(x) & = & R_{h_{min}} \cdot \frac{x_{max}}{x} & \text{(valve resistance)} \\
position & = & \text{crossed, if } i_A < i_B & \text{(valve position)}
\end{array}$$

$$\text{If crossed} \left\{ \begin{array}{lcll}
Q_P & = & -Q_B & \text{(continuity condition)} \\
p_P & = & p_B + sign(Q_P) \cdot R_h \cdot Q_P^2 & \text{(valve pressure drop)}
\end{array} \right.$$

Note that hydraulic components may have states that determine which part of their behavior description is actually valid.

The Checking Task

The following situation is the starting point of the problem of setting a system into operation: We are given a set of demands D and a hydraulic system C, and the task is to check if there are syntactical, geometrical, or behavioral faults. E.g., to identiy behavioral faults, the course of the forces and velocities of the cylinders or the admissible pressure values of crticial components have to be checked. If a fault is detected, the hydraulic system C must be modified. I.e., setting C into operation may constitute a sequence of checking and midification tasks, where, in turn, the checking task is comprised of the two subtasks *analysis* and *evaluation*. Figure 3.3 illustrates this cycle. Here, B_e denotes the expected behavior that can be derived canonically from D, and B_C denotes the behavior that is produced by the configured system C.

$$\begin{tikzcd}
D \sim B_e \arrow[rr, "Modification"] \arrow[rrd, leftrightarrow, "Evaluation"'] & & C \arrow[d, "Analysis"] \\
& & B_C
\end{tikzcd}$$

Figure 3.3: *An abstract view of setting a system into operation*

Let us take a closer look at the procedure of figure 3.3.

Analysis. Main job of the analysis step is the simulation of the hydraulic system C. In this connection, the engineer decides up to which level of detail the components' behavior must be modeled in order to obtain useful simulation results. While the stationary behavior is investigated always, the dynamic behavior is simulated for sensitive parts of the system only. Note that both cases are difficult from the mathematical standpoint since equation systems with non-linear and differential relations must be processed. Moreover, several assumptions about the components' states have to be met in order to set up a global behavior description. If a state assumption is wrong, the whole computation will fail and a new global description must be set up. Proposing useful assumptions needs both experience and a thorough investigation of the system's topology.

Evaluation. Within the evaluation step the analysis results are balanced with the demands D. Among others, the following questions must be answered: Does the switching logic realize the desired behavior? Will the piston velocities and forces be as prescribed? Are the maximum pressure values permissible? Do all geometrical connections fit? Will the velocity of flow allowed never be exceeded? Are the prescribed security and control measures considered?

Modification. Input for the modification step is the interpretation of the deviations found during the evaluation stage. E.g., if a hydraulic system has logical faults, the topology must be adapted or redesigned. Correcting dimensional faults means to select components of same type but with a different characteristic: valves, for instance, are exchanged with respect to their hydraulic resistance, cylinders with respect to their cross-section. After such a modification all computations have to be performed again.

Let us again consider the procedure in figure 3.3. An automation of the entire cycle is not possible because of the creativity that could be needed within the modification step $B_C \rightarrow C$. However, this modification step is usually not time-consuming for a human expert. Put another way, a reduction of just the analysis step's complexity would lead to a noticeable simplification of the entire process of setting C into operation.

Also note that automating the time-consuming checking tasks will allow human experts more room for creative jobs. Moreover, efficient checking concepts form the base for other tasks such as parameterization and optimization.

3.2 Analyzing Hydraulic Systems Analysis

As argued in the previous section, an automated analysis of hydraulic systems is the key for the checking of hydraulic systems. In this place we will investigate the analysis step in more detail.

From Local Behavior to Global Behavior

Loosely speaking, hydraulic systems analysis takes a circuit diagram as input and produces a behavior description of the entire circuit. For this, aside from the simulation job, also some kind of *model synthesis* problem has to be tackled.

By model synthesis we comprise all steps that are necessary to set up a model which is both correct in a physical sense and *locally* unique.

Note that even though a circuit diagram may establish a correct physical model, it is not locally unique as a rule: Each component of the circuit is defined by a set of behavior constraints from which the actually relevant ones must be selected. And, verifying the correctness of a local behavior description needs an expensive simulation of the *entire* system in most cases.

The indeterminacy of local behavior descriptions originates from the following reasons:

1. *Level of Description.* Each hydraulic component can be described at various levels. To avoid superfluous computational effort, an adequate level of detail, respecting both the rest of the circuit and the simulation intention, has to be determined for each component. On the other hand, we have to ensure that all components' descriptions fit together.

2. *Dynamical Simulation.* It must be analyzed which part(s) of the circuit require a dynamical investigation. For the possible components a stationary *and* a dynamic behavior model must be determined.

3. *Component States.* Most components have different physical states, each coupled with a particular behavior description. Which state the actually valid is depends on

the entire system and the actual input parameters. Example: A pressure relieve valve may be opened or closed.

4. *Topology.* A hydraulic system's topology can change with a component's state. Example: Dependent on its switching position a proportional valve connects different parts of a hydraulic network.

5. *Thresholds.* Even for a fixed state the direction or the absolute value of a physical quantity, which is a-priori unknown, may cause different behaviors of a component. Example: A turbulent flow is described by another pressure drop law than a laminar flow.

Tackling point 1 and 2 requires an engineers experience and heuristical knowledge in the first place. The points 3, 4 and 5 reveal that the analysis step $C \rightarrow B_C$ does also contain a selection step $C \rightarrow M_C$, where M_C denotes a set of behavior descriptions that are valid in the actual situation of the hydraulic system. Often, an additional cycle of model selection and model simulation is necessary to get a grip on this selection problem. Figure 3.4 illustrates the situation.

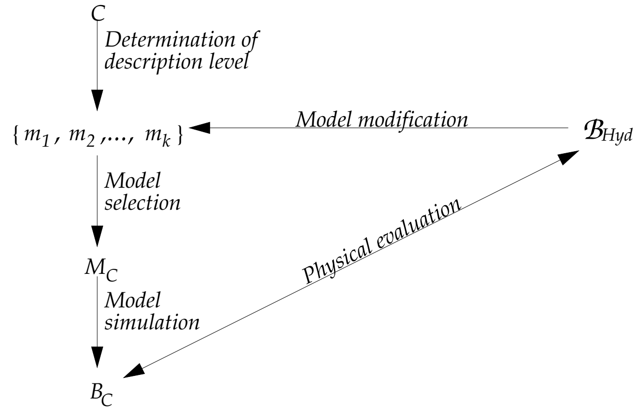


Figure 3.4: *What happens during the hydraulic analysis step*

Remarks. Let $\{m_1, \ldots, m_k\}$ comprise the behavior descriptions, say, models of all components at the adequate level according to point 1 and 2. From this set a subset is selected ($= M_C$), simulated ($\Rightarrow B_C$), and compared to $\mathcal{B}_{Hyd}$, which stands for the universal behavior laws of hydraulics. In the case that the simulated behavior B_C is physically contradictory or undetermined, M_C must be modified.

This cycle of selection, simulation, evaluation, and modification constitutes an inherently combinatorial problem; it is solved when the physically correct behavior descriptions according to the points 3 to 5 and $\mathcal{B}_{Hyd}$ are determined.

Note that this model synthesis problem is not treated explicitly in literature on the subject. Existing simulation tools leave the problem to the user who has to set up the correct equations and conditions respectively. The next subsection exemplary illustrates this statement.

Existing Tools

We found special-purpose and standard tools that support the analysis of hydraulic systems, e.g. MOSIHS [36], OHCS [34], MOBILE [21], or SIMULINK [33]. The majority of these tools has been developed to envision the dynamical behavior of (hydraulic) systems. So, typically configurational aspects like different checking or optimization tasks are addressed to a small part only. Another characteristic of such simulation tools is that they hardly support model synthesis. Subsequently, the efficiency that comes up with an automated model synthesis is pointed out at the commonly used simulation tool SIMULINK.

Modeling a system with SIMULINK requires the specification of a directed behavior graph whose nodes are mathematical functions; in fact, the complex transformation of a hydraulic circuit diagram into a mathematical description is left to the user. Figure 3.5 shows a simple circuit consisting of a cylinder and two hydraulic resistances. Its SIMULINK-counterpart is depicted in figure 3.6.

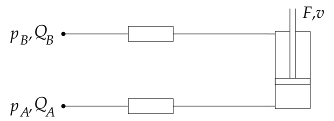


Figure 3.5: *Hydraulic specification of a simple circuit*

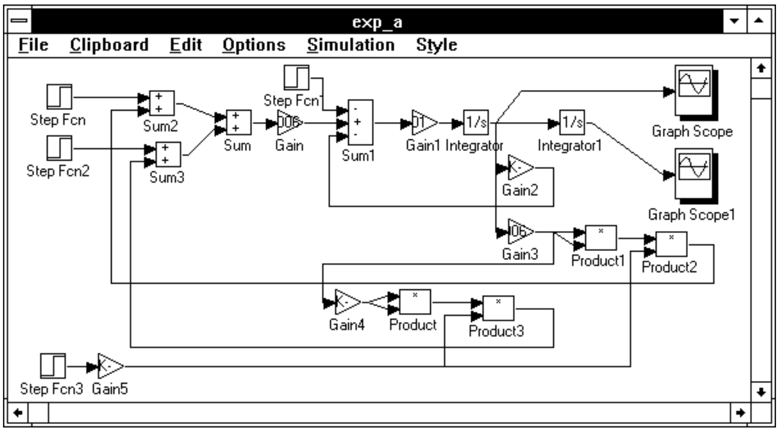


Figure 3.6: *Specification of the example in* SIMULINK

Remarks. Note that only the simpliest stationary dependencies are modeled in figure 3.6. Also note that if a user wanted to investigate the same circuit at some deeper level, he would have to start from scratch with the analysis process.

While the *circuit* diagram can be understood and created by every engineer, a description level similar to that in figure 3.6 requires a hydraulic specialist. Certainly, the building blocks of SIMULINK are flexible, but they are too simple to reduce the analysis step's complexity noticeable.

Discussion

An automated analysis step is a necessary condition for all kinds of configuration support such as automated checking, parameterization or optimization of hydraulic systems. The difficulty of the analysis step's automation comes up with the following points:

1. *Arbitrary Structures.* Hydraulic circuit analysis needs the processing of arbitrary structures and thus, some kind of model synthesis—that is: determination of the components' description level, selection of local behavior descriptions, and composition of the local descriptions to a global behavior model. By contrast, if all structures of hydraulic systems were already known, the corresponding global models could be precalculated.

2. *Definable Component Behavior.* The behavior of all existing and all components developed in future cannot be anticipated. Also, there is disagreement of how hydraulic components behave with respect to particular physical details. Thus, not only a hydraulic system's structure but also the component descriptions must be user-definable up to a point.

3. *Heterogeneous Constraints.* Behavior descriptions, user demands, necessary physical knowledge, heuristic design knowledge, etc. form a set of heterogeneous constraints that comprises several types of numerical and symbolic relations. These constraints, enclosed the dependencies between different types of constraints of course, must be both exactly formulated and processed.

3.3 Modeling Hydraulic Systems

In order to automate the formerly described checking task we need a modeling concept for hydraulic systems. This section presents a domain-oriented component model for hydraulic systems that, in particular, addresses the previously discussed aspects: modeling of arbitrary hydraulic structures, definable behavior at the *component level*, and coupling of heterogeneous constraints. Using this model, we are able to define precisely the hydraulic checking and parameterization problem.

A Component Model for Hydraulic Configuration

The "classical" approaches of description of technical systems are discussed by deKleer and Brown [9], Kippe [22], Kuipers [24], Strußs [42], or Voss [45]. Struß's approach is similar to that of Kippe's COMMODEL system. Both approaches are derivatives of deKleers's and Brown's modeling ideas that largely found on the locality-principle and the no-function-structure-principle[1] [9].

[1] Loosely speaking, a component will fulfill the no-function-structure-principle if its description never depends on its context of use.

The approaches mentioned have the following concepts in common: (*i*) They distinguish between components that realize the actual behavior and, for connection purposes, linking-components without behavior. (*ii*) The properties of the medium transported (e.g. electric current or oil viscosity) are modeled as an integral part of the components. Bound up with these concepts are some disadvantages concerning the structure and behavior definition of technical systems [18]. The approach introduced now is more flexible.

The modeling idea grounds on the distinction of objects, gates, unifiers, and the types of information transported. The unifiers serve as sources or sinks of the information types in a technical system; the objects are characterized by their behavior constraints and their gates. Each gate is connected to exactly one unifier. Note that objects will have access to the information of those unifiers connected to their gates. Also note that objects which share the same unifiers share the unifiers' information too.

Objects, gates, and unifiers define a system's topology and its possible flows of information. Figure 3.7 illustrates these dependencies, *Definition 3.1* formalizes them.

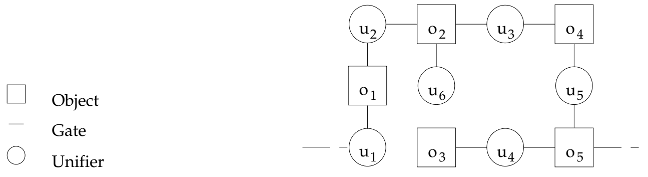


Figure 3.7: *Building block abstraction of a technical system*

Definition 3.1 (Building Block Model). A building block model is a tuple $\langle O, M, U, \gamma, \delta \rangle$ whose elements are defined as follows.

- O is an arbitrary finite set, it is called the object set.
- M is an arbitrary finite set, it is called the set of *information types*.
- U is a finite multiset of *unifiers*. Each $u \in U$ denotes an arbitrary subset of M, i.e., $u \subseteq M, u \in U$.
- $\gamma : O \rightarrow \mathbf{N}$ is a function and specifies for each object $o \in O$ the total number of gates.
- $\delta : O \times \mathbf{N} \rightarrow U$ is a partially defined function. $\delta(o, n)$, $o \in O$, $n \in \mathbf{N}$ specifies the unifier of object o at gate n and is defined for $n \leq \gamma(o)$ only.

Using these concepts, a hydraulic system is modeled as follows.

1. Each component of the system (valve, cylinder, pipe, etc.) is associated one-to-one with an element in O. $\gamma(o)$ defines a component's number of gates; e.g., if o is associated with a pipe then $\gamma(o) = 2$.
2. All physical quantities that have no manifestation within a component (pressure, flow, force, velocity, etc.) form the set M of information types.

3. δ is defined implicitly by the system's topology: For each link between two components in the hydraulic system a unifier with an appropriate subset of M is introduced. Example: If in the real system a certain pipe is connected with a particular valve fitting, in the building block modeling both the pipe's and the valve's fitting will share the same unifier u; u then ought to provide the information types "flow" and "pressure".

4. Each object is described by a set of behavior constraints that models the behavior of its associated component. An object's behavior constraints can refer to that information types only (e.g. a pressure or a flow) the object has access to via its gates. Note that same information types in different unifiers will establish different constraint parameters.

Figure 3.8 shows an example.

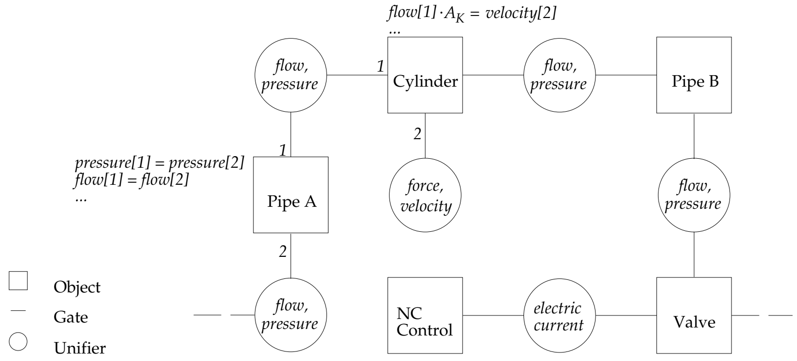


Figure 3.8: *Modeling a hydraulic circuit with the building block concept*

Until now we left open how behavior constraints for the objects in O can be formulated. This is made up now—we outline the key concepts of a language for behavior constraints that is tailored to the recently defined building block model.

- Relations, defined over numerical and symbolic parameters, are the basic elements of our constraint language. A relation may be defined as follows. (*i*) explicitly, in the form of a finite set of tuples; (*ii*) implicitly, in the form of a boolean statement or an equation, which is composed of numerical or symbolic expressions.

  Whereas boolean statements are treated as tests, the semantics of equations is handled also in a deductional manner: If an unknown parameter constitutes one side of an equation while the other side can be evaluated, the "="-sign will produce a parameter assignment. If both sides of an equation can be evaluated, the "="-sign will produce a boolean test. In all other cases an equation may be transformed according to algebraic or some other rules allowed for the constraint.

- A behavior constraint is usually associated with a particular object o. The parameters the constraint is defined upon refer to the information types at the gates of o and to the internal properties of o. The exact assignment of a parameter is indicated by its tag, which is either a gate specifier or the key word "self". E.g., the following algebraic constraint defines the pressure drop of a valve between gate 1 and gate 2 due to Bernoulli:

$$pressure[1] - pressure[2] = R_H[\textsc{Self}] \cdot flow[1]^2$$

- To each parameter a domain is defined. This domain is either an interval $v \subseteq \mathbf{R}$ or a finite set itemizing each single value allowed. Examples:

$$\begin{aligned} v_{valve\_resistance} &= [0.01,\ 0.2], \\ v_{valve\_position} &= \{\text{crossed, blocked, parallel}\} \end{aligned}$$

  The semantics of the latter is that, aside from *Unknown*, exactly one element of the specified set is admissible for the parameter $valve\_position$.

- Parts of an object's behavior description need to be activated or deactivated—e.g. to imitate the different states of a hydraulic component. A universal concept to realize such *model selection constraints* are rules. Thus, we allow behavior constraints to be embedded within rules. The condition part of a rule is a boolean statement composed of numerical and symbolic expressions; it specifies the conditions under which a behavior alternative is valid. The conclusion part defines the behavior constraints of the alternative and possibly additional rules. The following example shows a simplified pressure relief valve where the state "activated" is associated with a specific behavior:

$$\begin{array}{ll} \textsc{If} & state[\textsc{Self}] = \text{activated} \\ \textsc{Then} & Q[1] = -Q[2] \quad \textsc{And} \quad p[1] - p[2] = R_H[\textsc{Self}] \cdot \sqrt{|Q[1]|} \\ \textsc{Else} & Q[1] = 0 \quad \textsc{And} \quad Q[2] = 0 \end{array}$$

  The constraints in the conclusion part will be considered during behavior processing only if the condition part evaluates to *True*.

- Behavior constraints can be supplied with metaknowledge that specifies hints to be exploited during the behavior processing. This knowledge may indicate a constraint's description level, or whether a constraint contains stationary or dynamic dependencies, or some other processing directive. Table 3.1 gives some examples.

Constraint	**Metaknowledge**
$p[1] - p[2] = R_H[\textsc{Self}] \cdot Q[1]^2$	level-0 description, stationary behavior
$Q[1] + Q[2] + Q[3] = 0$	level-independent description, process locally

Table 3.1: *Metaknowledge specifications for behavior constraints*

Remarks. The building block model of *Definition 3.1* along with the outlined behavior language make up our component model for the configuration of hydraulic systems.

Instances of Π^{c} in Hydraulics

Remember the behavior-based configuration problems defined in chapter 2. Obviously, the component model above is a *device-oriented interpretation* of such a behavior-based model: It represents the *engineer's* view of configuration in the sense that it introduces domain concepts, defines a semantics, and is oriented by knowledge acquisition purposes.

Hence, hydraulic analyzing, checking, and parameterization problems are particular instances of the generic behavior-based checking problem Π^{c}, defined on page 9.

More precisely—a behavior-based checking problem (Π^{c}) is defined by the tuple $\langle O$, F, V, B, T, D, $S\rangle$. These elements are related to our component model as follows.

- O is equivalent to the objects of the building block model and denotes the hydraulic components.
- All parameters in the objects' behavior constraints form the set F of functionalities. Each element $f \in F$ denotes a (possibly constant) function in the parameter "time".
- V comprises all functionalities' value sets.
- The set B of behavior constraints comprises the behavior descriptions of the hydraulic components at some definite level.
- T comprises all functionalities' tests.
- D is the set of demands on the hydraulic system, stated by a customer.
- The pairs in S correspond to the elements in the unifiers $u, u \in U$ and define the structure of a hydraulic system.

Remarks. The checking problem Π^{c} in hydraulics defines a hydraulic system and a set of demands. Checking this system means both determining a set Q of tuples (f, x), $f \in F$, $x \in v_f$, which are consistent with all behavior constraints, and verifying if none of the demands is violated.[2] These jobs are done during the hydraulic analysis and evaluation step respectively.

Tackling a *parameterization* problem means to solve the checking problem Π^{c} for a hydraulic system that is underspecified within one constraint or other.

3.4 Solving Π^{c} in Hydraulics

This section introduces the necessary concepts to solve instances of Π^{c} in hydraulics. Figure 3.9 shows the steps that are performed when analyzing, checking, or parameterizing a hydraulic system.

[2] A precise definition is given in section 2, page 9.

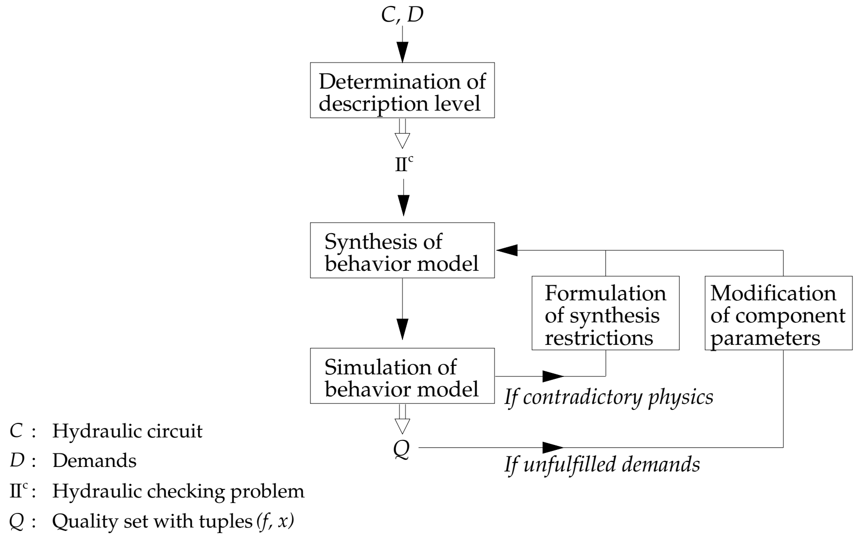

C, D
Determination of description level
Π^c
Synthesis of behavior model
Formulation of synthesis restrictions
Modification of component parameters
Simulation of behavior model
If contradictory physics
Q
If unfulfilled demands
C : Hydraulic circuit
D : Demands
Π^c : Hydraulic checking problem
Q : Quality set with tuples (f, x)

Figure 3.9: *Solving* Π^c *in hydraulics*

Remarks. Given is a hydraulic system C for which, in a first step, an adequate description level has to be determined. Based on this modeling of C and the set D of demands, an instance of Π^c can be stated. Within the subsequent synthesis and simulation steps the set Q of functionality-value pairs, which explicitly envisions the behavior of C, is inferred. Q constitutes a solution of Π^c if both all behavior constraints and all demands can be fulfilled.

Determination of the Description Level

The components of a hydraulic system can be described at different levels of detail. To avoid superfluous computational effort, a component's behavior description should be as simple as possible—but sufficiently detailed yet to model all necessary relations.

To get a grip on this model formulation problem we developed, together with Lemmen [28] and Suermann [43], the concept of variable hydraulic modeling levels. In particular, Lemmen defined a model formulation scheme that, dependent on a component class, distinguishes up to five predefined description levels. Given a hydraulic system C, then for each component the adequate description level can be inferred by means of a knowledge-based decision procedure that bases on the natural frequencies and gain factors of the components in C, a user's simulation intention, and the topology of C. A detailed description of the model formulation scheme and the decision procedure can be found in [28] and [43].

Since the determination of the adequate description level is a hydraulic engineering problem in a high degree it shall not be extended here. Henceforth, we assume all components of a hydraulic system described at some definite level; analyzing, checking, or parameterizing such a system is an instance of Π^c.

Synthesis and Simulation of Behavior Models

Given is an instance of Π^{c} in hydraulics. Π^{c} defines a constraint satisfaction problem consisting of numerical and symbolic relations. Actually, Π^{c} cannot be solved by a universal "constraint satisfaction algorithm" but needs the interplay of several computation methods, and a global control mechanism that separates and triggers sub-jobs, maintains alternatives, and controls model synthesis.

As argued before, model synthesis is not a deterministic procedure here; there exist choice points where, dependent on the actual input values, parameter alternatives, or physical regularities the valid component model must be selected: For each component $o \in O$, O defined by Π^{c}, let $M_o = \{m_{o_1}, m_{o_2}, \ldots, m_{o_k}\}$ comprise the k_o behavior alternatives of o. If a component o has a locally unique model, say, a pipe for instance, $|M_o| = 1$. Let $\mathcal{M}_C$ be the Cartesian product of the $M_o, o \in O$. Obviously, $\mathcal{M}_C$ comprises the possible global models of the hydraulic system C defined by Π^{c} and thus, $\mathcal{M}_C$ defines the total synthesis search space.

Before all physical parameters of a hydraulic system C can be computed, a physically consistent model $M_C \in \mathcal{M}_C$ has to be determined. Conversely, whether a behavior model M_C is physically consistent can solely be verified via simulation. To illustrate the search for a consistent behavior model in $\mathcal{M}_C$, it is useful to think of the components' behavior constraints being divided into model selection constraints (cf. page 20) and behavior constraints. The search procedure can be outlined as a cycle comprising the following inference steps:

1. *Component Selection.* Select a component with undetermined behavior.

2. *Model Selection.* Select a behavior alternative for this component.

3. *Synthesis.* Identify and evaluate active model selection constraints, and synthesize the emergent behavior model.

4. *Simulation.* Simulate the synthesized behavior model by evaluating the behavior constraints.

5. *Model Modification.* In case of physical inconsistencies or unfulfilled demands formulate synthesis restrictions and trace back to a choice point.

Figure 3.10 illustrates the search process graphically.

The search comes to an end if either a global, consistent behavior model is found that fulfills all demands or no further choice point exists.

Remarks. Different components constrain the model synthesis step in a different manner. Hence, the order by which undetermined components are processed may play a crucial role.

The evaluation of behavior constraints, mentioned in the simulation step of the above search procedure is a demanding problem: The components' functional constraints must be parsed, symbolic relations must be separated from numerical relations, equations have to be transformed, equation systems have to be formulated in some normal form, rules

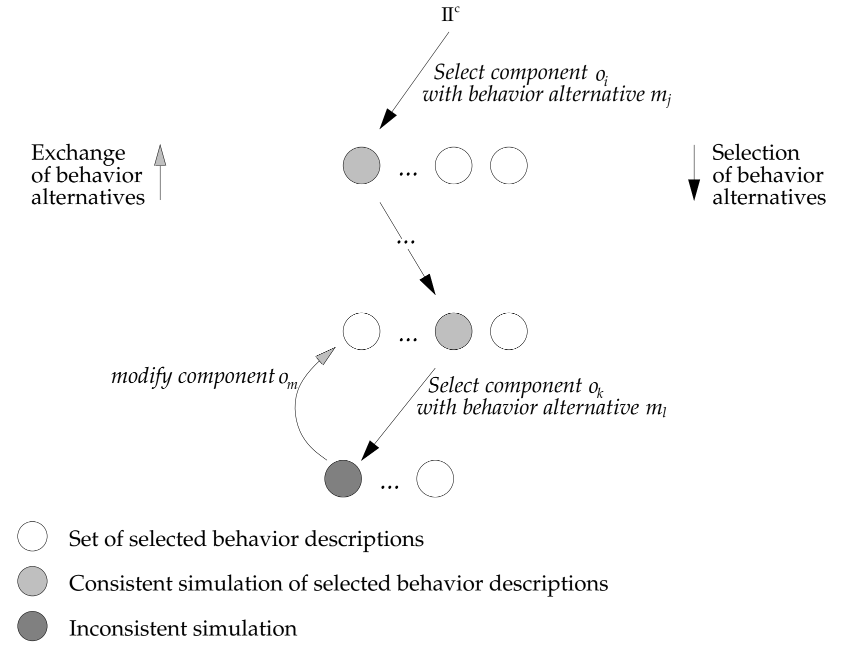

Π^c
Select component o_i with behavior alternative m_j
Exchange of behavior alternatives
Selection of behavior alternatives
...
modify component o_m
Select component o_k with behavior alternative m_l
Set of selected behavior descriptions
Consistent simulation of selected behavior descriptions
Inconsistent simulation

Figure 3.10: *Exploring the synthesis search space $\mathcal{M}_C$*

have to be processed, etc. Moreover, through the variety of constraint types and constraint dependencies, the backtracking mentioned in the model modification step will become a sophisticated job, too. In this place, we will not go into constraint processing details but merely give an overview of the inference methods necessary to solve Π^c.

Necessary Inference Methods

Following numerical sub-jobs must be performed when processing Π^c in hydraulics:

- *Solving Linear Equation Systems.* Input is a linear equation system in the matrix form $A \cdot x + b = 0$ where A is regular. Output is the vector x that solves the equation system.

- *Solving Non-Linear Equation Systems.* Input is a continuous, non-linear function $f : \mathbf{R}^n \to \mathbf{R}^n$ and a possibly empty set of restrictions constraining the value ranges of the solution in $m \leq n$ dimensions. Output is a vector x that fulfills both the equation $f(x) = 0$ and all restrictions.

- *Solving Initial Value Problems.* Input is a system of ordinary differential equations $y' = f(x, y)$ where $f : G \to \mathbf{R}^n$ is a continuous function defined on $G = [a, b] \times \mathbf{R}^n$, and an initial condition $\alpha \in \mathbf{R}^n$. Output is a function $y = u(x)$ that solves the differential equation system and fulfills the initial condition $u(a) = \alpha$.

Algorithms that process the itemized problems are given in [39]. Aside from methods coping with numerical problems, we need the following methods for symbolic value processing:

- *Local Value Propagation.* Input are the sets F (functionalities), V (value domains), B (behavior constraints) as described on page 21, and an initial value assignment X. Each behavior constraint $b \in B_o$ defines a relation on $v_{f_{b_1}} \times v_{f_{b_2}} \times \ldots \times v_{f_{b_k}}$ where $f_{b_i} \in F$, $\{b_1, b_2, \ldots, b_k\} \subseteq \{1, 2, \ldots, n\}$ and $n = |F|$. X defines a tuple $(x_1, x_2, \ldots, x_n)$, $x_i \in v_i \cup \{\text{unknown}\}$, $v_i \in V$ where some or all elements may be unknown.

  By local value propagation we designate a deduction mechanism that exploits only one constraint definition in B at the same time to compute values for the unknowns in X. An example for such a mechanism is the following:

  1. Select a behavior constraint $b \in B$ where all parameters except one are known. If no such constraint exists, local propagation comes to an end. Otherwise, without loss of generality, we assume x_j to be the unknown parameter.
  2. Determine a value for $x_j \in v_j$ such that the relation defined by b is fulfilled and all behavior constraints also defined on v_j stay consistent. If no such x_j can be determined but earlier choice points exist, invoke backtracking. Otherwise, local propagation terminates and $\langle B, X \rangle$ is called inconsistent.
  3. Continue with 1.

  Remarks. The algorithm computes a globally consistent solution if one exists. Nevertheless, dependent on the constraints and the domains they are defined upon, several approaches for local propagation and consistency definitions exist on which shall not be elaborated here.
  Constraints may be defined *extensionally*, by a complete enumeration of the relations' tuples, or *implicitly* by a set of functions. In the latter case, the power and flexibility of local propagation depends on the algorithms that realize the evaluation of the functions. Note that the principle of local constraint evaluation can be applied to any type of relation.

- *Rule Inference.* Rule inference in the form of forward chaining is required to evaluate selection constraints, functional constraints, and demand constraints. Note that not only plain symbolic relations but also dependencies between different sets of behavior constraints must be handled.

- *Algebraic Transformation.* Algebraic transformation capabilities are necessary to process the implicitly defined constraints and to generate the input forms for the numerical computation methods.

3.5 Preprocessing of Stationary Behavior

Processing stationary behavior constraints requires methods capable of coping with equation systems, local propagation, and algebraic transformation. The approach presented now substitutes the computation of *local* connections with that of *global* connections. The key idea is to preprocess the topology of a hydraulic system: global structure information is "compiled" into local behavior constraints, which are added to the original constraints. The resulting component descriptions can be processed by local propagation, which is more efficient than the solution of equation systems necessary for the original descriptions.

Motivation

A large part of a hydraulic system can be considered as a network consisting of resistances, sources, and sinks. Pipes and valves establish the hydraulic resistances (R), while pumps, tanks, and—of course—cylinders act as sources (s) and sinks (t) respectively. Figure 3.11 depicts an example.

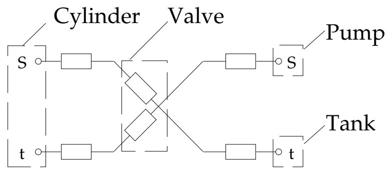


Figure 3.11: *An abstract view on a simple circuit*

A central task of the hydraulic checking problem Π^c is the computation of a total flow distribution and of all pressure drops for such a network. As a difference to electrical resistors, hydraulic resistances define non-linear connections—more exactly: in most cases the potential difference $p_1 - p_2$ at a hydraulic resistance is proportional to the quadratic flow:

$$p_1 - p_2 = R_h \cdot sign(Q) \cdot Q^2$$

The previous section introduced local propagation as a method that evaluates *single* behavior constraints of components in order to compute unknown functionalities. This method is very efficient since no global relations are exploited but will terminate, if no constraint can be found where all parameters except one are known. E.g., if two hydraulic resistances are connected in parallel, the computation of the flow distribution and the pressure drops often requires the solution of a non-linear equation system. Figure 3.12 gives examples of structures that may be part of a hydraulic resistance network.

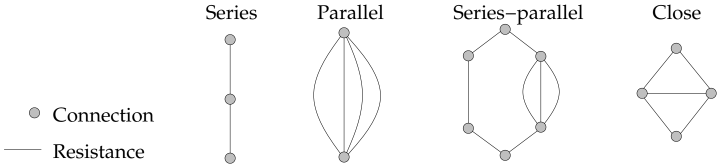


Figure 3.12: *Possible substructures in a resistance network*

The edges denote hydraulic resistances while the points stand for components where all incident resistances are connected (unified). Such components might be T-connections or other connections that establish an area of equal potential. Within a context-free modeling, the only behavior constraint of a connection is the *continuity condition*:

$$Q_1 + Q_2 + \ldots + Q_n = 0$$

Preprocessing means the introduction of additional constraints, called *proportion constraints* here. Proportion constraints are computed from the resistances between a source and a sink. They are installed in the connections and provide information about how the flow Q is distributed—example:

$$Q_1 = c_1 \cdot Q_e \ \wedge \ Q_2 = c_2 \cdot Q_e \ \wedge \ Q_1 + Q_2 = Q_e, \ c_1, c_2 \in \mathbf{R}^+$$

Q_e is introduced as a new variable and denotes the entire flow at a connection, c_1 and c_2 determine how this flow is distributed, and Q_e is substituted for $Q_1 + Q_2$ in the original continuity condition. Obviously, by aid of proportion constraints, local value propagation will be sufficient to distribute the flow at the connections and to compute all related pressure drops, if at a network source (e.g. at a pump) a flow is given. Of course, proportion constraints violate the no-function-in-structure-principle since they are not context-free: They exploit global information about resistances and the network's structure.

The following subsections show how those structures of a network that cannot be tackled by local propagation are found and how the related proportion constraints are computed.

Finding All Relevant Substructures

In principle, proportion constraints could be computed for a network's global structure in a single computation step. But for flexibility reasons, it is much more appropriate to cut a network into subnetworks and to compute proportion constraints locally for each substructure: The smaller a separated subnetwork is the more flexible the computed proportion constraints can be used.

Proportion constraints can be computed only for subnetworks of a particular structure. Now we will define these structures and present methods to identify them as parts of a global network. We will use the following definitions of graph theory in the standard way:

1. A *multigraph* G is a triple $\langle V, E, g\rangle$ where $V, E \neq \emptyset$ are finite sets[3], $V \cap E = \emptyset$, and $g : E \to 2^V$ is a mapping with $2^V = \{U \mid U \subseteq V, \ |U| = 2\}$. V is called the set of points, E is called the set of edges, and g is called the incidence map.

2. A graph $H = \langle V_H, E_H, g_H\rangle$ will be called *subgraph* of $G = \langle V, E, g\rangle$, if $V_H \subseteq V$, $E_H \subseteq E$, and g_H is the restriction of g to E_H. A subgraph will be called an *induced subgraph* on V_H, if $E_H \subseteq E$ contains exactly those edges incident to the points in V_H. For $T \subset V$, $G \setminus T$ denotes the subgraph induced on $V \setminus T$.

3. A tuple $(e_1, \ldots, e_n)$ will be called a walk from v_0 to v_n, if $g(e_i) = \{v_{i-1}, v_i\}$, $v_i \in V$, $i = 1, \ldots, n$. G will be called *connected*, if for each two points $v_i, v_j \in V$ there is a walk from v_i to v_j. If G is connected and $G \setminus v$ is not connected, v establishes an *articulation point*.

4. $\kappa(G)$ is called the *connectivity* of G and is defined as follows: $\kappa(G) = min\{|T| : T \subset V \text{ and } G \setminus T \text{ is not connected}\}$. G is called *k-connected*, if $\kappa(G) \geq k$.

Figure 3.13 illustrates the definitions.

[3] We restrict ourselves to finite graphs here.

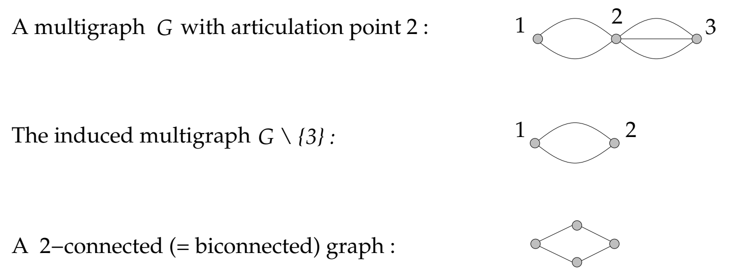

A multigraph G with articulation point 2 :
1
2
3
The induced multigraph G \ {3} :
1
2
A 2–connected (= biconnected) graph :

Figure 3.13: *Sample graphs*

The edges of a multigraph stand for the hydraulic resistances such as pipes and valves, the points connect the resistors and represent areas of equal potential. We need multigraphs instead of graphs here since components of a hydraulic system may be connected in parallel. Moreover, we shall restrict to particular multigraphs, the so-called resistance networks:

Definition 3.2 (Resistance Network). A *resistance network* N is a tuple $\langle G, \rho, s, t \rangle$ where $G = \langle V, E, g \rangle$ is a connected multigraph, $\rho : E \to \mathbf{R}^+$ is a mapping, and $s, t \in V$ are two points. $\rho(e), e \in E$ is called the resistance of e; s, t are called the source and the sink respectively.

Remarks. In contrast to "flow networks" or "capacitated networks" that define capacity values for the edges of a graph, ρ defines resistance values in the physical sense of hydraulics. Resistance networks and capacitated networks are used in connection with *flows*. We refrain from a precise definition of flows but shall point out three important characteristics:

1. A flow defines both a flow value and a flow direction on the edges of a resistance network.
2. To each point $v \in V \setminus \{s, t\}$ applies the continuity condition: The total of all input flows equals the total of all output flows.
3. All incident edges of s (of t) establish output (input) flows only. The total of s's output flows equals the total of t's input flows. The latter is a direct consequence of 2.

Note that a flow distribution in a physical sense is nothing to do with a flow mapping under a maximum-capacity interpretation of ρ. In the former case, a flow distribution depends on the resistance *ratios* of *all* edges. In the latter case, all edges' capacity values must be considered as well, but the capacities are independent of their actual context in the graph: Every edge can be checked *locally* whether its related flow violates the edge's capacity.

We are interested in those parts of a network[4] whose flow distribution can be computed independently of the rest. For obvious, physical reasons, each subnetwork whose resistance behavior can be reproduced by a substitute resistance—i.e., by a single edge—constitutes such a part. Following definitions will be useful:

[4] Henceforth, we shall refer to a "resistance network" simplified as "network".

Definition 3.3 (Independent Subnetwork). Let be $N = \langle G, \rho, s, t\rangle$ a network, H a subgraph of G induced on $V_H \subset V$ with $|V_H| > 2$, and ρ_H the restriction of ρ to E_H. Then, a network $N_H = \langle H, \rho_H, s_H, t_H\rangle$ will be called an *independent subnetwork* of N, if the following conditions hold:

(*i*) Every walk from s (from t) to a point in H contains either s_H or t_H.

(*ii*) Every walk from a point in $G \setminus V_H$ to a point in H contains either s_H or t_H.

An independent subnetwork $N_{H_1} = \langle H_1, \rho_{H_1}, s_{H_1}, t_{H_1}\rangle$ will be called *minimum independent subnetwork* of N, if there exists no independent subnetwork $N_{H_2} = \langle H_2, \rho_{H_2}, s_{H_2}, t_{H_2}\rangle$ where H_2 is induced on a proper subset of V_{H_1}.

Remarks. Since any two adjacent points establish a source and a sink respectively, we claim H to be defined on more than two points. Condition (*i*) guarantees some kind of "dipole character" of H. Condition (*ii*) ensures that there are exactly two connections for a substitute resistance. Note that this is not implied by (*i*) . Also note that proportion constraints computed for minimum independent subnetworks provide the maximum flexibility in the course of local propagation.

Before we turn our attention to the question as to how minimum independent subnetworks are detected, we need a further definition:

Definition 3.4 (Triconnected Component, Biconnecting Points). Let $G = \langle V, E, g\rangle$ be a multigraph, $|V| > 2$, $\kappa(G) = 2$, and $\{v_i, v_j\} \in V$ two points such that $H = G \setminus \{v_i, v_j\}$ is not connected. Moreover, let be H_1 a resulting connected component of H and $V_{H_1} \subset V$ the set of points inducing H_1.

If no two points $\{w_i, w_j\} \in V$ can be found such that a resulting connected component of $G \setminus \{w_i, w_j\}$ is induced on a proper subset of V_{H_1}, then the graph H_2 induced on $V_{H_1} \cup \{v_i, v_j\}$ shall be called a *triconnected component* of G. The points v_i, v_j shall be called the *biconnecting points* of H_2 related to G.

Figure 3.14 gives an example.

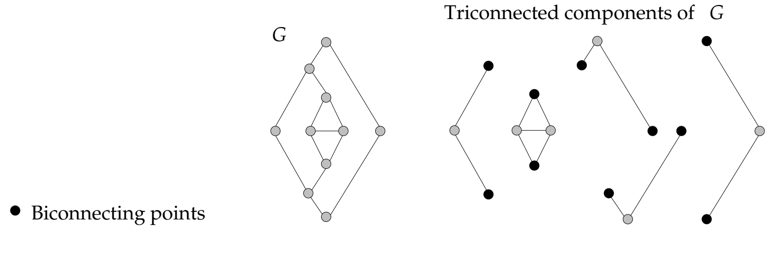


Figure 3.14: *Examples for triconnected components and biconnecting points*

Lemma 3.5 (Independent Subnetwork). Let be $N = \langle G, \rho, s, t\rangle$ a network and $|V| > 2$.

1. If $\kappa(G) = 1$, let $G_1, \ldots, G_n$ denote the biconnected components of G. Then, if $|V_i| > 2$, there exist two points $s_i, t_i \in V_i$ such that $N = \langle G_i, \rho_i, s_i, t_i\rangle$ is an independent subnetwork of N, $i \in \{1, \ldots, n\}$.

2. If $\kappa(G) = 2$, let denote $G_1, \ldots, G_n$ the triconnected components of G and s_i, t_i the biconnecting points of G_i related to G. Then, $N_i = \langle G_i, \rho_i, s_i, t_i \rangle$ will be a minimum independent subnetwork of N, if $V_i \setminus \{s_i, t_i\} \cap \{s, t\} = \emptyset$, $i \in \{1, \ldots, n\}$.

3. If $\kappa(G) > 2$, N will not contain an independent subnetwork.

Proof. (1) Let G_i be a biconnected component. We have to investigate two cases: V_i contains either one or two articulation points of G. Case A: v is the only articulation point in V_i. Then, either s or t must be in V_i. If $s \in V_i$, set $s_i = s$ and $t_i = v$; if $t \in V_i$, set $s_i = v$ and $t_i = t$. Case B: Let v_i, v_j be the articulation points in V_i. Then, $s_i = v_i$ and $t_i = v_j$. We can check easily that $\langle G_i, \rho_i, s_i, t_i \rangle$ fulfills the definition of an independent subnetwork.

(2) Let G_i be a triconnected component. Part one (independent subnetwork): According to the definition of triconnected components, a walk from any point in $G \setminus G_i$ to a point in G_i contains either s_i or t_i (this is a consequence of the biconnectivity of G). Thus, condition (*ii*) of the independent subnetwork definition is fulfilled. Also, condition (*ii*) together with the restriction that $V_i \setminus \{s_i, t_i\} \cap \{s, t\} = \emptyset$ fulfill part (*i*) of the independent subnetwork definition. Part two (minimality): We assume N_i not to be minimum. Then, there exists an independent subnetwork $H = \langle H, \rho_H, s_H, t_H \rangle$ with $H = G_i \setminus T, T \subset V_i$. I.e., G_i cannot be a triconnected component. This contradicts the condition.

(3) Is obvious since the $\kappa(G) \leq 2$ is a direct consequence of the independent subnetwork definition. ◇

Remarks. The independent subnetworks found for $\kappa(G) = 1$ are not necessarily minimum. They might be processed further until the connectivity of their related graph is > 1. The complexity for the computation of all minimum independent subnetworks of a network N with a graph $G = \langle V, E, g \rangle$ can be estimated with $O(|V| \cdot |E|)$. We outline only the proof idea. All triconnected components of a graph G with $\kappa(G) > 1$ can be found as follows. A point $v \in V$ is selected and the graph H, induced on $V \setminus \{v\}$, is investigated with regard to its biconnected components. Obviously, the biconnected components of H are triconnected components of G, if they cannot be extended by v, the point initially removed. For an articulation point w, found during the biconnected component search, the graph induced on $V \setminus \{w\}$ needs not to be investigated. In worst case $\kappa(G) > 2$, and for each $v \in V$ the induced graph is investigated with regard to biconnected components. According to Tarjan, the biconnected components of a graph G can be computed in $O(|E|)$ [44].

In practice, the identification of independent subnetworks will be less complex: Often, there exist particular structures, which can be detected easily, the so-called series-parallel networks. Also, for physical reasons, the computation of substitute resistances for series-parallel networks is much easier than for networks relying on close-connected graphs. Hoffmann provides a more detailed discussion of this topic [18].

Installing Proportion Constraints

Subsequently, we describe the general installation procedure of proportion constraints in a network $N = \langle G, \rho, s, t \rangle$. Initially, $N_2 = \langle G_2, \rho_2, s_2, t_2 \rangle$ is a copy of N.

1. Identify an independent subnetwork $N_H = \langle H, \rho_H, s_H, t_H \rangle$ of N_2 .
2. Compute the flow distribution of N_H.
3. Install the proportion constraints in the original graph G.
4. Condensate G_2, i.e., replace H by the new edge e_H and redefine ρ_2.
5. Continue with 1 until s_2 and t_2 are the only points of G_2.

Remarks. Before close connected subgraphs are treated, all series-parallel structures should be searched and replaced. Note that due to the condensation of G_2, new series-parallel and close connected structures may emerge. Figure 3.15 illustrates the process.

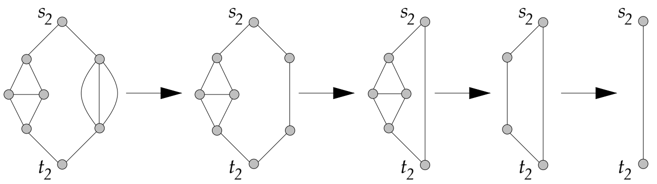


Figure 3.15: *Condensation of G_2 during constraint computation*

The computation of substitute resistances and flow distributions for N_H with graph $H = \langle V_H, E_H, g_H \rangle$ bases on the continuity conditions in V_H and the pressure drop equations instantiated for each $e \in E_H$:

- *H is Series Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 are restricted to E_2 where $\rho_2(e_H) := \sum \rho_H(e_i)$, $e_i \in E_H$.

- *H is Parallel Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 is restricted to E_2 where $\rho_2(e_H)$ can be computed from the following equation:
$$\sqrt[2]{\frac{1}{\rho_2(e_H)}} = \sum \sqrt[2]{\frac{1}{\rho_H(e_i)}},\ e_i \in E_H$$
In s_H and t_H, the following proportion constraints are installed:
$$Q_i := c_i \cdot Q_H,\ \ c_i := \sqrt[2]{\frac{\rho_2(e_H)}{\rho_H(e_i)}},\ \ e_i \in E_H,\ \text{and } Q_H = \sum Q_i$$

- *H is Close Connected.* N_2 with graph $G_2 = \langle V_2, E_2, g_2 \rangle$ is modified as follows. $V_2 := V_2 \setminus V_H \cup \{s_H, t_H\}$, $E_2 := E_2 \setminus E_H \cup \{e_H\}$, g_2 and ρ_2 is restricted to E_2 where $\rho_2(e_H)$ can be computed from the following connection: The continuity conditions of all points $\in V_H \setminus \{s_H, t_H\}$ along with the pressure drop equations for each $e \in E_H$ and the equation $p_s - p_t = x$, $x \in \mathbf{R}^+$ form a non-linear equation system. Under the restriction that all pressure drops are positive, it can be shown that this equation system has a definite solution in the flows Q_i, $i = 1, \ldots, |E_H|$. Then, $Q_H := \sum Q_i$, e_i is incident to s_H (t_H), establishes the total flow through H. As a result, $\rho_2(e_H) := x \cdot Q_H^{-2}$. Now, a proportion constraint can be formulated for each flow variable: $Q_i = c_i \cdot Q_H$, $e_i \in E_H$ where the c_i are computed from the solutions of the equation system: $c_i := Q_i \cdot Q_H^{-1}$. These proportion constraints and the equation $Q_H := \sum Q_i,\ \ e_i$ is incident to s_H (to t_H) are installed in s_H (in t_H).

Discussion

The above preprocessing approach focuses on optimization tasks where the structure of a complex hydraulic system is still defined but several alternative situations need to be investigated and evaluated. It will not be useful for the investigation of small hydraulic systems or a single simulation.

Chapter 4

The $^{\text{art}}$*deco* System

$^{\text{art}}$*deco* is a system that supports the configuration of hydraulic systems. It operationalizes the component model and a large part of the concepts presented in the former chapter. Until now, $^{\text{art}}$*deco* solves several instances of Π^c, that is, the analysis and the checking of hydraulic systems, and the parameterization of single component parameters [23], [29], [40], [41].

However, solving instances of Π^c is not enough to support a user in configuring hydraulic systems: When supporting configuration in hydraulics, aside from a knowledge processing task, there is also a *problem specification* and a *knowledge acquisition* task to be tackled.

By the term "problem specification" we denote the procedure of formulating an instance of Π^c in hydraulics, in other words: How can a user specify his problem in an acceptable time?—Knowledge acquisition is of equal importance: Configuration support in hydraulics will be useless, if a user cannot integrate his individual knowledge, his experience in component modeling, or specifications of new components. Both aspects were taken into account when developing $^{\text{art}}$*deco*. Besides efficient inference concepts, $^{\text{art}}$*deco* realizes *graphic problem specification* and provides a language to model component behavior.

This chapter introduces the philosophy and some concepts of $^{\text{art}}$*deco* but does not engage in the details of realizational aspects.

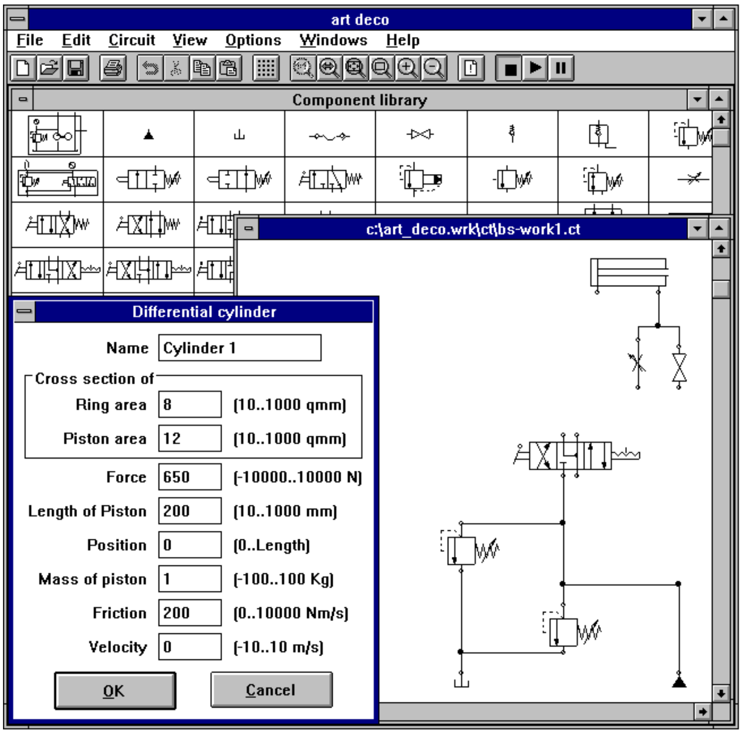
4.1 Graphic Problem Specification

A configuration process that is grounded in behavior descriptions is usually so complex that its complete automation is not possible. Then, the job of a configuration system is not to *solve* but rather to *support* the creative design process. When given such a behavior-based configuration problem, technical dependencies might be so complicated that problem specification, knowledge acquisition, and maintenance can solely be understood on a very abstract level, e.g., on the level of a technical drawing.

This is the situation when designing hydraulic systems, and at this point the philosophy of $^{\text{art}}$*deco* comes into play:

The working document of the design process is the circuit diagram. Consequently, it would be fair to specify hydraulic checking problems at the same level of abstraction. Graphic symbols should be selected and connected to a circuit—but, in contrast to a CAD system, aside from the drawing, a *functional model* of the hydraulic system should be generated as well.

artdeco realizes such graphic problem specification: While the circuit diagram of a system is drawn, a knowledge base containing all necessary physical connections is created. Within a second step, arbitrary sections of the hydraulic system can be checked concerning individual demands. I.e., the model composition/formulation process as well as complex physical dependencies are made transparent: Nearly all information obligatory for the checking and simulation process is derived from the technical drawing. Brought down to a simple formula, *artdeco* is CAD + behavioral semantics.



Name: Cylinder 1

Cross section of

Ring area: 8 [10..1000 qmm]

Piston area: 12 [10..1000 qmm]

Force: 650 [-10000..10000 N]

Length of Piston: 200 [10..1000 mm]

Position: 0 [0..Length]

Mass of piston: 1 [-100..100 Kg]

Friction: 200 [0..10000 Nm/s]

Velocity: 0 [-10..10 m/s]

Figure 4.1: *artdeco's application mode*

Specifying Hydraulic Problems with *artdeco*

There is always a gap between the semantics of a configuration problem on the one hand and the syntax for its specification on the other. *artdeco* addresses this situation by expunging the gap between the process of drawing and the process of investigating/simulating a technical system; it can be considered as a visual language to specify a particular class of technical problems.

Being in *artdeco*'s application mode, a user selects hydraulic components from a component catalog and arranges them on the working area (cf. figure 4.1).

While drawing a line between two components' gates, the appropriate pipes are selected and instantiated. Among other things, it is checked whether the incident gates are of same type. The necessary information concerning the topology is generated as well. Within the circuit diagram, all parameters of the hydraulic system can be predefined, changed, or supplied with alternative values.

After the inference process is invoked, ^{art}*deco*'s inference engine searches for a consistent parameter assignment. Figure 4.2 shows a circuit where such an assignment has been found.

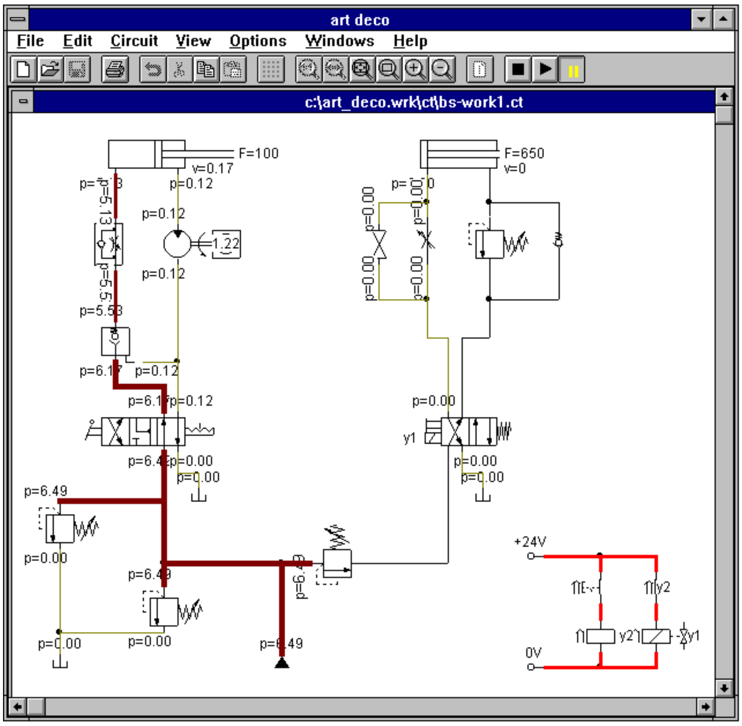


Figure 4.2: ^{art}*deco's application mode*

The prototypic use of ^{art}*deco*has shown that its philosophy of graphic problem specification leads to a decisive reduction of the specification complexity. Maybe, graphic problem specification as realized here is the only chance to efficiently support complex configuration tasks in hydraulics.

4.2 Inference

^{art}*deco* takes the graphical description of a hydraulic circuit and investigates the system with respect to the following faults:

1. *syntactical* faults like open pipes
2. *geometrical* faults like wrong connections
3. *logical* faults like piston movements contradicting to valve positions

4. *dimensional* faults like pumps whose power range is exceeded

If none of these faults can be found, all components' states are determined, and, along with any unknown velocities, forces, and component parameters an entire distribution of the flow and the pressure is computed.

The subsequent paragraphs outline the underlying inference process.

Constraint Processing at First Glance

From the standpoint of problem specification and knowledge processing we distinguish between the following constraint classes in ^{art}*deco*:

- *Connection Constraints.* Connection constraints establish if and how two components may be connected. Processing these constraints means to check all connections' types and port sizes, the mechanical couplings, and for open pipes. Since this checking step is both the least demanding one and separable from other constraint processing jobs it is always performed first.
- *Topological Constraints.* Topological constraints are given by the circuit diagram and define the physical structure of the hydraulic system. A correct realization of this structure is achieved as follows: All local component parameters are replaced by global variables, which in turn are unified according to the connection information. This unification step takes place before the functional constraints are processed.
- *Behavior Constraints.* Behavior constraints define the local behavior of components and are user-definable. They consist of relations, defined over numerical and symbolic parameters. Parameters can be constrained through a domain; the constraints themselves can be supplied with metaknowledge.
- *Model Selection Constraints.* Model selection constraints are used to define different component states; they associate a state description with a particular behavior alternative. A behavior alternative comprises a set of behavior constraints.
- *Demand Constraints.* Demand constraints comprise internal and external restrictions. Internal demand constraints check for violations of universal behavior laws of hydraulics; external demand constraints model a user's demands and are specified in the form of rules and simple relations. In contrast to behavior constraints, the demand constraints are propagated *destructively*. I.e., they are not exploited to derive new dependencies but rather to check them. To perform *early pruning* in the course of constraint processing, demand constraints are checked after each inference step.

While a user is drawing a circuit diagram, a building block model of the hydraulic system is constructed (cf. page 18). When the inference process is started ^{art}*deco* processes the connection constraints and the topological constraints as defined by the building block model and formulates an instance of Π^c. This instance of Π^c in turn is processed in a cycle of model synthesis and model simulation (cf. figure 4.3 and figure 3.9 on page 22 respectively).

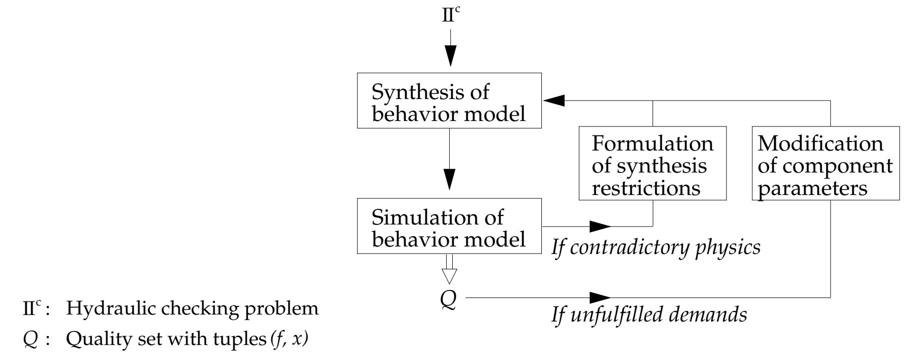

Π^c
Synthesis of behavior model
Formulation of synthesis restrictions
Modification of component parameters
Simulation of behavior model
If contradictory physics
Q
If unfulfilled demands
Π^c : Hydraulic checking problem
Q : Quality set with tuples (f, x)

Figure 4.3: *Solving* Π^c *in hydraulics*

Efficient Model Synthesis

Section 3.4 introduced the idea of a synthesis search space $\mathcal{M}_C$, which forms the set of possible global behavior models for a given system C. Actually, section 3.4 left open how a consistent behavior model—i.e., a behavior model that fulfills all behavior constraints and all demands—can be searched efficiently.

Note that even for a rather simple circuit, $\mathcal{M}_C$ might contain several thousand elements. And, checking the consistency of an element $M_C \in \mathcal{M}_C$ usually requires the simulation of M_C. Thus, an intelligent exploration of the synthesis search space $\mathcal{M}_C$ is the key factor which decides whether a solution of Π^c can be found at all in an acceptable time.

To get a grip on this model synthesis problem we developed domain-independent and domain-dependent concepts to control the exploration of $\mathcal{M}_C$. The essentials of our concepts are outlined below.

- *Incremental Constraint Update.* Each time a new value is inferred, its side effects on the model selection constraints are computed immediately.
- *Dependency Recording.* Within each inference step, the inferred values are labeled with the responsible assumptions. The dependency recording in *artdeco* establishes cause-effect links between single parameters as well as between different sets of constraints, regardless of their type.
- *Topological Analysis.* The topology of a hydraulic circuit is investigated in order to determine global dependencies that have an effect on the constraint selection (e.g. flow direction analysis).
- *Domain Heuristics.* Domain heuristics that define preferences on behavior alternatives are evaluated during the inference process.

These concepts are tied together to a model synthesis control, which makes up a large part of *artdeco*'s inference engine. Figure 4.4 and 4.5 show *artdeco*'s entire inference procedure at an abstract level.

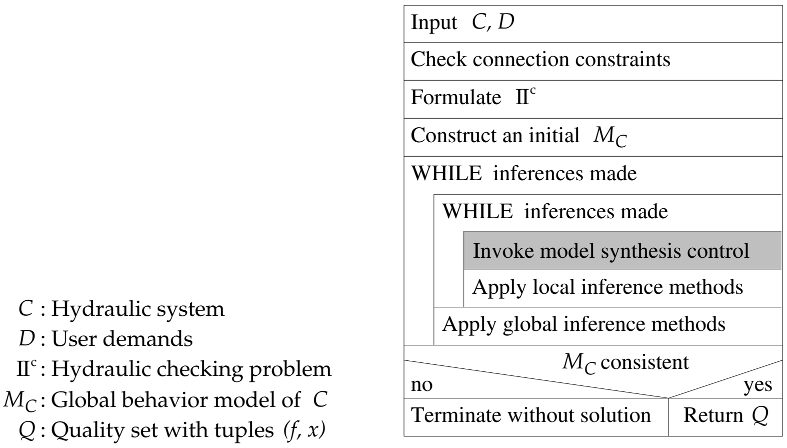

Input C, D
Check connection constraints
Formulate Π^c
Construct an initial M_C
WHILE inferences made
WHILE inferences made
Invoke model synthesis control
Apply local inference methods
Apply global inference methods
M_C consistent
no
yes
Terminate without solution
Return Q
C : Hydraulic system
D : User demands
Π^c : Hydraulic checking problem
M_C : Global behavior model of C
Q : Quality set with tuples (f, x)

Figure 4.4: *Main inference loop*

Remarks. Within the main inference loop we distinguish local and global inference methods. The former class comprises the methods for local value propagation, rule inference, and algebraic transformation; the methods of the latter class handle different types of equation systems. Section 3.4 gave an overview of these methods. Since local inference is causal and often more efficient as compared to global inference, it is applied before any global inference method is tried.

To ensure early pruning in the course of the exploration of $\mathcal{M}_C$, the model synthesis control is invoked after each inference step.

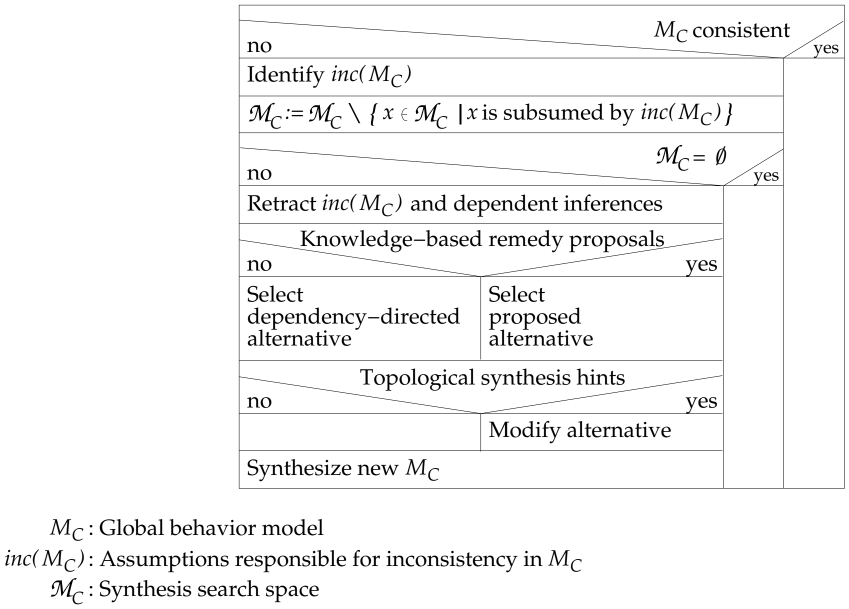

M_C consistent
no
yes
Identify $inc(M_C)$
$\mathcal{M}_C := \mathcal{M}_C \setminus \{ x \in \mathcal{M}_C \mid x$ is subsumed by $inc(M_C) \}$
$\mathcal{M}_C = \emptyset$
no
yes
Retract $inc(M_C)$ and dependent inferences
Knowledge–based remedy proposals
no
yes
Select dependency–directed alternative
Select proposed alternative
Topological synthesis hints
no
yes
Modify alternative
Synthesize new M_C
M_C : Global behavior model
$inc(M_C)$: Assumptions responsible for inconsistency in M_C
$\mathcal{M}_C$: Synthesis search space

Figure 4.5: *Model synthesis control*

Remarks. The set $\{x \in \mathcal{M}_C | x$ is subsumed by $inc(M_C)\}$ in figure 4.5 comprises those global behavior models of C whose assumptions contain the set $inc(M_C)$.

Note that the alternative selection and modification controls model synthesis by means of dependency-directed backtracking, knowledge-based backtracking, and the evaluation of topology constraints.

The identification of the assumptions $inc(M_C)$, which are responsible for an inconsistency in M_C—as well as the retraction of all consequences involved, requires a fairly sophisticated dependency recording.

Dependency Recording

The constraint network established by Π^c consists of two kinds of nodes: nodes referring to functionalities (parameters) and nodes referring to behavior constraints. A constraint node b and a functionality node f will be linked by an edge, if f stands in the relation defined by b. Initially, each functionality is labeled either by f's alternative states or by *Unknown* (cf. figure 4.6).

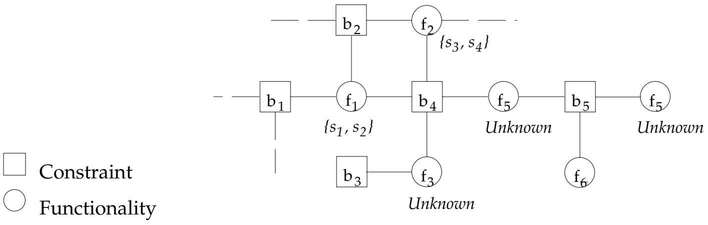


Figure 4.6: *The constraint network defined by the checking problem*

The value sets of the nodes f labeled *Unknown* are considered to be restricted by v_f. What happens during the constraint satisfaction process is that step by step constraints are evaluated and value sets are cut down to those values that match (fulfill) all actually triggered constraints. If a non-empty value set is assigned to each functionality, and each tuple $(x_1, x_2, \ldots, x_n)$, induced by these value sets, fulfills all constraints $b \in B$, the constraint satisfaction problem will be solved.

If a contradiction occurs in the course of constraint processing the responsible nodes must be detected, and these nodes with all their consequences must be retracted. Then, an alternative value assignment of the nodes that caused the contradiction can be selected and inference can be continued.

Dependency recording must be tailored to both the inference mechanisms and the constraints. Here, we distinguish between three types of dependency links:

- *Constraint Dependency.* Constraints can depend directly on other constraints, the so-called model selection constraints. If such a constraint is fulfilled, the dependent constraints are "active"; otherwise, they are "inactive". Throughout constraint processing, new links are introduced in the constraint network that indicate both the inference (= supports) and the retract (= depends) direction (cf. figure 4.7).

- *Local Value Dependency.* The constraints processed during local propagation define a cause-effect chain between the nodes of the network. These relationships are recorded by the introduction of support links and by node labeling. So, the root nodes

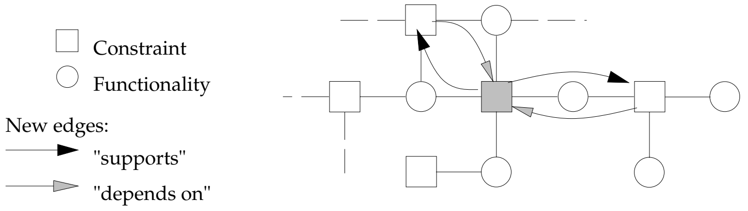


Figure 4.7: *Dependency recording between constraints*

of an inconsistency can be determined immediately, and their consequences can be traced back (cf. figure 4.8).

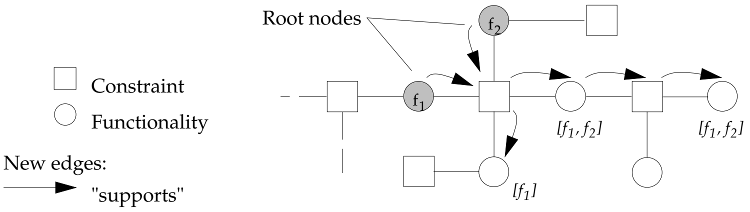


Figure 4.8: *Dependency recording during local propagation*

- *Global Value Dependency.* Constraints that cannot be treated by local propagation are called global. Global constraints establish cyclic dependencies between functionalities and constraints. Retracting one node of such a strong connected component results in the retraction of all nodes involved. In order to avoid labeling effort in $O(n \cdot m)$, with n, m specifying the number of functionalities and constraints respectively, not all dependency links are installed in the graph but those that are necessary to instantiate a strongly connected component.

This dependency recording concept is an integrated part of each inference method in $^{\text{art}}$*deco*. It forms the base for the *dependency-directed* and the *knowledge-based* backtracking: In either strategy, the setting back to arbitrary points of the inference process is necessary. The former sets back to the root nodes of a contradiction where a new value assignment (= new alternative) is chosen chronologically. The latter provides deeper information about which of the alternatives of the root nodes should be modified. In both cases all inferences that are no longer supported must be retracted, i.e., the constraint network has to be relabeled.

Discussion

There is a lot of research related to “constraint satisfaction problems” [5], [11], [14], [16], [17], but only a small part of this research actually contributes to hydraulic configuration. The term “constraint satisfaction problem” is somewhat misleading here since it is a label used for very different problem classes: One part of these problems is tackled by some kind of constraint propagation, while another part needs some kind of inherently global inference. E.g. Davis mentions six different categories of constraint propagation that are distinguished by the type of information which is updated [5].

An important category of constraint satisfaction problems are the so-called *label inference* problems. They deal with a network of nodes, each labeled with a set of possible values, and constraints that are used to restrict the value sets. Related to such problems, and certainly useful, are the terms “node consistency”, “arc consistency”, and “path consistency”, which define different levels of local consistency [14], [32].

Note that in our constraint satisfaction problem local consistency is not crucial. Most of the constraints define underdetermined relations on infinite sets. I.e., filtering value sets in its classical sense is hardly possible—but, filtering in the form of value range propagation is useful and partly necessary: value range information is exploited during the numerical subjobs and the constraint selection process.

Rather than label inference the following inference types are employed to tackle the hydraulic checking problem:

- *Constraint Inference.* Constraint inference denotes a process where new constraints are inferred and added to the network.
- *Value Inference.* In the course of value inference, initially labeled nodes (the assumptions) along with the constraints are used to infer values of unlabeled nodes.

To make things worse, our constraint satisfaction problem Π^{c} is inhomogeneous, i.e., very different types of constraints are employed. Thus, it cannot be tackled by a single method but needs a global control mechanism that both combines all required computation methods and maintains dependencies.

The dependency management in $^{\text{art}}$*deco* adopts concepts from Doyle’s justification-based truth maintenance system (JTMS) [12] and from deKleer’s assumption-based truth maintenance system (ATMS) [7]. Employing the classical ATMS-based dependency management would not be useful for performance reasons here: (*i*) Label-inferencing and updating all combinations of assumptions is not necessary, and (*ii*) maintaining ATMS-data structures poses an overhead as compared to recording the cause-effect dependencies during local propagation.

4.3 Knowledge Acquisition

$^{\text{art}}$*deco* operationalizes the component model of section 3.3. This component model defines all $^{\text{art}}$*deco* object classes, the structure of these classes, the building block model, and the syntax of the behavior descriptions. These concepts are an integral part of $^{\text{art}}$*deco*’s philosophy: They are intended to simplify the creation and the processing of new components and cannot be modified.

Most of these concepts are kept transparent: From the users’ point of view, $^{\text{art}}$*deco* objects represent a data structure that defines physical and graphic information (cf. figure 4.9).

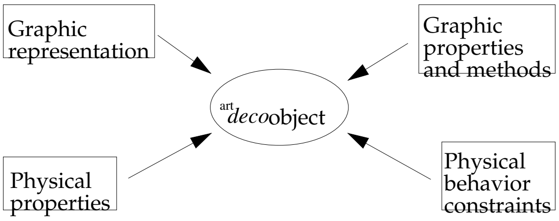

Graphic representation
$^{\text{art}}$*deco*object
Graphic properties and methods
Physical properties
Physical behavior constraints

Figure 4.9: *The user's view of $^{\text{art}}$deco objects*

Physical and graphic information need to be "synchronized". More exactly: If a connection line between two objects is drawn, it has to be ensured that there is also a physical correspondence between these objects. This correspondence is established by the gate concept: Gates are designated areas of an objects graphic representation. They define how external physical parameters can be referred and where components can be connected graphically.

Following EBNF-notation describes those parts of an $^{\text{art}}$*deco*-object that are user-definable:

$\langle\text{component}\rangle \longrightarrow \langle\text{name}\rangle\langle\text{physical-representation}\rangle\langle\text{graphic-representation}\rangle$
$\langle\text{physical-representation}\rangle \longrightarrow$
$\quad \{\text{GATE}.\langle\text{number}\rangle\}_1^* \; \{\langle\text{functionality}\rangle\}_0^* \; \{\langle\text{behavior-constraint}\rangle\}_0^*$
$\langle\text{functionality}\rangle \longrightarrow \langle\text{name}\rangle\langle\text{value}\rangle\langle\text{default}\rangle\langle\text{alternatives}\rangle$
$\langle\text{name}\rangle \longrightarrow \langle\text{symbol}\rangle$
$\langle\text{default}\rangle \longrightarrow \langle\text{value}\rangle$
$\langle\text{alternatives}\rangle \longrightarrow (\{\langle\text{value}\rangle\}_0^*)$
$\langle\text{value}\rangle \longrightarrow \langle\text{symbol}\rangle \mid \langle\text{number}\rangle$

Remarks. The term ⟨behavior-constraint⟩ denotes an expression in $^{\text{art}}$*deco*'s behavior description language; ⟨graphic-representation⟩ denotes a collection of graphic primitives with different mouse-sensitive regions.

If a user wants to create a new component, he has to provide a set of behavior constraints as well as a graphic description with designated gates. $^{\text{art}}$*deco* takes this information, instantiates the necessary objects, updates the component model, and converts the behavior description into an internal form.

Strategy Language

$^{\text{art}}$*deco*'s global inference strategy can be redefined. More exactly: $^{\text{art}}$*deco* provides a set of atomic inference techniques coping with different types of constraints like symbolic relations, equation systems, etc. Using some kind of BNF-syntax, these deduction techniques can be composed easily to a new, individual inference strategy. This is useful for adapting the inference process to the type of constraints given or to particularities of the domain.

The EBNF-notation below defines all productions that form a valid strategy for $^{\text{art}}$*deco*'s inference process. In this connection, the semantics of the $\oplus$-symbol is as follows. The preceding deduction technique will be repeated until no further inference can be drawn. Note that this situation is always reached after a finite number of steps, since the number of constraints, variables, and alternatives is finite, and cyclic dependencies are detected.

$$\begin{array}{lcl}
\langle\text{control-strategy}\rangle & \longrightarrow & \{\langle\text{control-strategy}\rangle\}_0^* \mid (\langle\text{control-strategy}\rangle)\oplus \mid \\
 & & \langle\text{local-inference}\rangle \mid \langle\text{global-inference}\rangle \\
\langle\text{local-inference}\rangle & \longrightarrow & \textsc{Local-Numeric-Propagation} \mid \\
 & & \textsc{Local-Symbolic-Propagation} \mid \\
 & & \textsc{Conditional-Constraint-Propagation} \mid \\
 & & \textsc{Demand-Test} \\
\langle\text{global-inference}\rangle & \longrightarrow & \textsc{Solve-Linear-Equation-System} \mid \\
 & & \textsc{Solve-Non-Linear-Equation-System} \mid \\
 & & \textsc{Solve-Differential-Equation-System}
\end{array}$$

Behavior Description Language

The behavior description language establishes the interface to component behavior. Via this language interface one is able to modify and to maintain behavior descriptions of existing $^{\text{art}}$*deco* objects as well as to define the behavior of new ones.

$^{\text{art}}$*deco*'s behavior description language is an implementation of the constraint language presented in section 3.3. Since the typical hydraulic engineer has no programming skill it is kept as simple as possible and free of programming language details. Some of its characteristics are:

- The language allows the formulation of (mixed) numerical and symbolic relations.
- The language is tailored to the connection philosophy of the building block model, which is defined on page 18. I.e., using the keywords [SELF] and [GATE], components may refer to their own functionalities as well as to information types at their gates.
- Parameters (variables) need not to be typed.

Usually, behavior constraints are specified in an *external* form, which in turn is converted into an internal representation that can be processed more efficiently. A detailed description of the constraint language and the converter can be found in [19].

4.4 Realization

Figure 4.10 gives an overview of $^{\text{art}}$*deco*'s main modules. Important modules are briefly described below.

User Interface. The user interface consists of three main parts: (*i*) The front end realizing the circuit drawing area where a user manipulates graphic symbols and dialog boxes, (*ii*) an action interpreter taking a user's actions (mouse drag, double-click, keyboard input, etc.) and, dependent on the context, decides whether an action is valid or not, and (*iii*) a module providing graphic routines for a CAD-like handling of circuit diagrams.

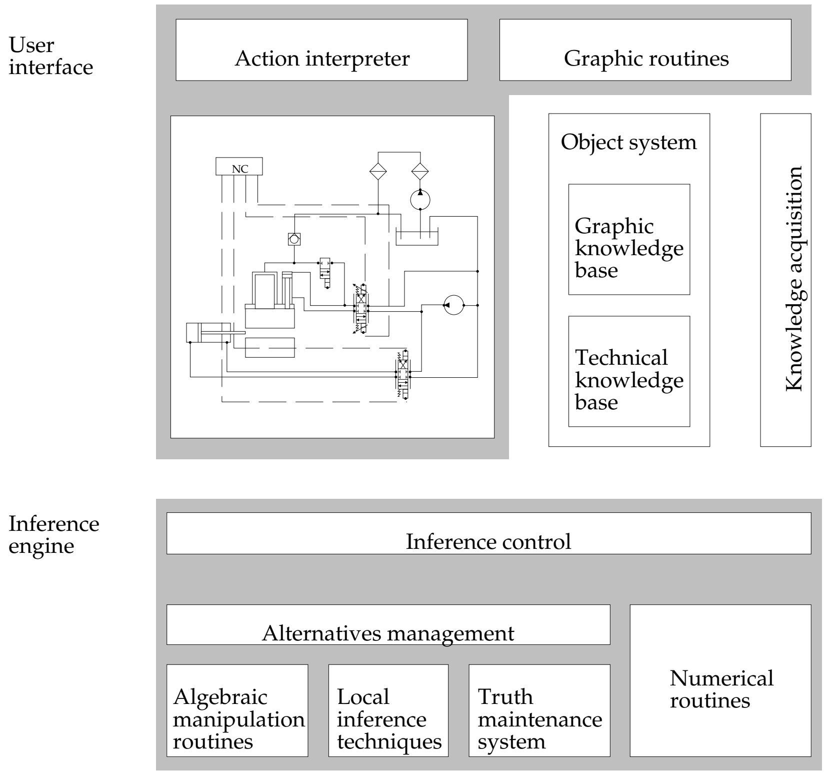


Figure 4.10: *Modules of ^{art}deco*

Knowledge Bases. *^{art}deco* provides a graphic and a technical knowledge base containing classes from which all user-visible objects are instantiated. The graphic classes predefine the eligible manipulation methods; the technical knowledge base defines the basic structure of important hydraulic component classes. The component catalog is built on top of these knowledge bases and can be extended with the aid of the acquisition module.

Knowledge Acquisition Module. The knowledge acquisition module provides the language for behavior descriptions outlined in the previous section and an interface for the import of new graphic descriptions. The module checks new descriptions with respect to different syntactical and semantic aspects, instantiates the necessary objects in the knowledge bases, and makes the new components available in the component catalog.

Inference Engine. The inference engine provides several local and global inference techniques that are invoked by a global inference control. The inference control can be imagined as an interpreter that takes a hydraulic circuit along with the technical knowledge base (= Π^c) as input and evaluates the global control strategy in order to find a solution of Π^c. For clarity reasons, the interplay between the different submodules is presented simplistically here.

Developmental Issues

There are two extreme positions of how a system that solves instances of Π^c in hydraulics may be developed:

1. *Tool-based.* By this approach we designate the strategy of selecting and combining tools where each solves a particular job of the entire problem: (*i*) MAPLE or MATHEMATICA, for instance, are employed to do the algebraic manipulation and numerical computation jobs, (*ii*) flexible object-oriented representation of data, rule processing, local propagation algorithms, and truth maintenance mechanisms are realized with aid of a powerful knowledge engineering tool, (*iii*) a CAD system establishes the front end, and (*iv*) all tools are controlled by a command language, e.g. by TCL/TK [35]. Additionally, we need algorithms that filter the graphic descriptions and formulate instances of Π^c which can be processed by the other tools.

2. *Language-based.* We will designate an approach language-based, if all concepts and algorithms are developed from scratch using one or more programming languages.

At first glance, the favorable developing approach seems to be closer to (1) than to (2). But for the following reasons rather the opposite is true:

- A circuit diagram given in a CAD system needs to be translated into single object representations that are both related one-to-one to hydraulic components and supplied with technical parameters and behavior descriptions.

- The integration of domain knowledge into a CAD system is difficult. But such an integration is exactly that what we need here: User decisions that are not permitted should be detected earliest possible in order to avoid superfluous simulation effort.

- Since none of the tools could the entire constraint processing, a common constraint representation needs to be developed. This representation must be supplied with a truth maintenance mechanism and a global inference control that triggers the computation jobs.

- Generic numerical routines do not exploit physical restrictions of the domain. As a consequence, they may be less efficient than specially adapted algorithms. Also note that a numerically correct solution needs not to be physically correct.

- Rule processing and constraint inference have to be developed and integrated within the common constraint representation. Since knowledge acquisition is a heterogeneous task here its operationalization would benefit from the language-based approach, too.

Tackling all these problems is a demanding job that cannot be done in a single step since new concepts need to be developed and evaluated. Moreover, users should participate in the development process as early as possible.

We addressed this situation by dividing the development process of $\overset{\text{art}}{\textit{deco}}$ into two stages:

1. *Prototype Stage.* In the first place, we had to get a clear idea of how configuration in hydraulics could be supported at all. Hand in hand went research related to graphic

problem formulation, the necessary inference types, the expressiveness of a component language, adequate data structures, and the interplay of different inference mechanisms.

2. *Reimplementation Stage.* The reimplementation stage was *not* a copy of the prototype stage. Rather, research has been concentrating on the design of *efficient* concepts and algorithms: From a user's point of view, a configuration problem cannot be "solved in principle" but needs to be solved in an acceptable time.

Throughout the prototype stage, we used the knowledge engineering environment KEE to realize our ideas [20]. At the end of this stage, $^{\mathrm{art}}$*deco*'s philosophy, its architecture, and a large part of the necessary methods were developed or evaluated. The algorithms (local deduction techniques, algebraic routines, graphic routines, the acquisition module) were written in COMMON LISP, the knowledge bases were built on top of the KEE object system, the user interface based on the KEE picture system. Aside from improving and developing our concepts, the KEE/LISP version of $^{\mathrm{art}}$*deco* served as a realistic communication base between users and developers.

When we started reimplementing $^{\mathrm{art}}$*deco*, we refrained from the employment of particular shells or knowledge engineering tools. Tools often restrict the portability and usually lead to a loss of performance.

$^{\mathrm{art}}$*deco*'s philosophy requires a tight combination of the inference process on the one hand and the user interaction on the other. Thus, we developed a small graphics kernel that can be ported easily to other platforms. On top of this graphics kernel we built a "semantic" graphics layer that provides powerful graphic commands *related to the application*. This semantic graphics layer interprets user actions and manipulates the technical and the graphic knowledge base. For efficiency and maintenance reasons, also a specialized object system for the representation of knowledge bases was developed.

If an inference process is invoked, $^{\mathrm{art}}$*deco* constructs a constraint satisfaction problem using the actual instantiations of the technical and the graphic knowledge base. This job is passed to the inference control that employs local and global inference techniques, truth maintenance, and an alternatives management to solve the problem. Since these techniques need a coordinated interplay both a generic constraint representation wherein all constraints can be formulated, and tailored constraint processing methods have been developed.

The actual version of $^{\mathrm{art}}$*deco* is realized in COMMON LISP and C/C++. The object system, the graphic interface, and the numerical routines are written in C/C++; the other modules of the inference engine are written in COMMON LISP. Parts of the knowledge acquisition module were developed with the UNIX tools LEX and YACC. At the present time, $^{\mathrm{art}}$*deco* runs on WINDOWS 3.1.

Chapter 5

Diagnosis of Hydraulic Systems

5.1 Introduction

Before we present concepts concerning the diagnosis of hydraulic systems, we will have a short look at troubleshooting aspects that relate to hydraulics. The following points should be noted.

- *Large Number of Sources of Faults.* The misbehavior of a hydraulic system may result from both broken components and structural faults. Structural faults lead to additional or missing connections between the components of a hydraulic system.
- *Multiple Faults.* By multiple faults we designate the presence of more than one structural fault or broken component at the same time.
- *Little Knowledge about Symptoms.* When the diagnosis of a malfunctioning hydraulic system starts, only one symptom is usually known. Often a single symptom is insufficient to perform a definite diagnosis; i.e., further investigations of the system must be carried out.
- *Difficult Measurements.* Measurements in hydraulic systems are normally incomplete, inexact, and expensive. Thus, the number of measurements should be kept at a minimum.
- *Qualitative Observations.* Observations of misbehavior need not be stated in the form of numerical values. For an engineer who is observing a hydraulic system it can be much easier to describe a symptom qualitatively. Example: “The cylinder’s piston is driving out too slowly.”
- *Knowledge about Faulty Behavior.* Experts have knowledge about the behavior of defect components. This knowledge is not part of a standard component description and hence, it has to be explicitly specified within a component’s behavior definition.

When investigating the aspects above, it becomes clear that a diagnosis concept should rather be model-based instead of heuristic[1]. Note that by the term “model-based” no par-

[1] A short review of the different approaches for diagnosis can be found in [37].

ticular diagnosis mechanism or algorithm is specified at all. This term does merely imply that the diagnosis process relies on balancing the simulated behavior of a model with the observed behavior of a real system. Using the model-based approach there is the challenge to master the following problems:

- At which level of abstraction should the model be formulated?
- How can the model be specified?
- By which mechanisms can the model be processed?

Our research showed that the component description that we used to perform the checking tasks is not adequate respecting diagnosis jobs. The reasons for this insight can be summarized as follows. The description covers only the correct behavior of components; secondly, it is too detailed with respect to particular concepts of typical hydraulic faults.

Possible component faults are listed in [27]. Moreover, ibid., it is proposed to separate between a global model-based diagnosis concept and a local heuristic diagnosis concept. With the global diagnosis process a hydraulic circuit is investigated at the circuit level, i.e., faulty components are detected. The local diagnosis process is ought to refine this result: It explains exactly which defect has been occurred within the faulty component.

This hierarchical concept imitates the diagnosis procedure of a human expert, who also applies some top-down strategy. The advantages of such a procedure are the following:

1. The model-based diagnosis process can be restricted to relatively simple behavior descriptions. Because it does not encode very particular component defects, it needs not to be adapted respecting new hydraulic components.
2. The heuristic diagnosis process at the component level works independently of its model-based counterpart. It can be easily enriched with an expert's individual diagnosis experience.

The model-based diagnosis process has been prototypic realized. The subsequent sections introduce the concepts.

5.2 Model-based Diagnosis

Model-based diagnosis grounds on a model that represents the interesting system's structure and behavior. Such a model allows the prediction of the behavior of the real system. The deviations found when comparing the predicted behavior of the simulated model to the observed behavior of the real system are called *symptoms*. These symptoms have to be explained in terms of one or several misbehaving components.

A set of misbehaving components that can explain all observed symptoms is called a *candidate*. Note that for a set of symptoms there may be more than one explanation, and thus, more than one candidate. In order to differentiate between several candidates, additional observations of the real system's behavior must be made.

The power of model-based diagnosis results from its modularity. Knowledge about faults needs not to be anticipated but can be derived from the system's structure + local component descriptions + simulation. A system whose components are modeled correctly will produce a correct global behavior as well.[2] Or conversely, each set of correctly modeled components whose predicted behavior does not correspond to the observed behavior contains at least one component that is faulty. Such a set of components is called a *conflict*.

Model-based diagnosis is comprised of the following 5 subtasks: model formulation, behavior prediction, conflict identification, candidate generation, and candidate discrimination. The subtasks 2 to 5 form the diagnosis process of the "General Diagnostic Engine" (GDE); the model formulation job is considered to be completed when diagnosis starts and is usually not counted to the diagnosis process.

The General Diagnostic Engine

This section introduces the fundamentals of the GDE related to a small hydraulic diagnosis example. The GDE is a domain-independent diagnosis mechanism, presented by deKleer and Williams [10]. It requires a component-oriented description of a system; component behavior is modeled by local constraints that define relations between a components' input and output parameters; behavior simulation is realized by constraint propagation. Moreover, each component is tagged with an assumption indicating whether the component does behave correctly or not. These correctness assumptions are considered when processing behavior constraints.

Figure 5.1 shows a hydraulic system that consists of two connected pipes. Let us consider that the incoming flow is about $6\, l/min$ at point A.

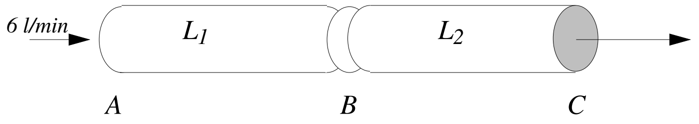


Figure 5.1: *Hydraulic circuit example consisting of two pipes*

As stated above, the GDE performs a cycle comprised of the following steps:

1. *Behavior Prediction.* In our example the expected flow at point C can be computed in the model via a propagation of the pipes' local behavior descriptions. It will also be $6\, l/min$ if we presume that the two pipes L_1 and L_2 work correctly. Each locally predicted value has a *label*, a particular set, whose elements in turn are sets containing components. The semantics is as follows. When given a particular system configuration, for instance the two pipes and an input flow, the (correctly working) components in each element of a label form a necessary and complete set to infer the value associated to this label. If there are, for example, two possibilities to infer a

[2] This concept is sometimes referred to as the "no-function-in-structure-principle". Loosely speaking, the no-function-in-structure-principle says that a component's behavior is independent of its context of use [4], [9], [24].

value, the associated label will contain two sets. Figure 5.2 shows the inferred flow values with their associated labels.

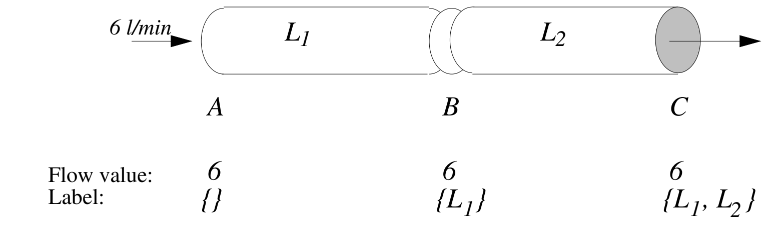


Figure 5.2: *Behavior prediction of the flow at the points B and C*

The elements of a label are called *environments* for the associated value. The minimum environment for some value is the empty set.

Note that behavior prediction in a technical system must not follow the physical propagation direction, which is directed from a component in-ports to its out-ports.

2. *Conflict Identification.* We now assume that at point C of the real system a flow value of $4\,l/min$ has been observed or measured. Based on this observation, the values and labels as shown in figure 5.3 can be stated by the GDE.

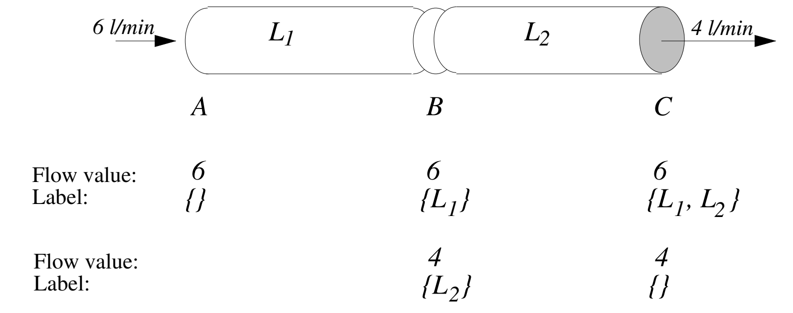


Figure 5.3: *Behavior prediction of the flow at the points B and C*

The flow at point B is predicted with $6\,l/min$ if pipe L_1 is assumed to work correctly but with $4\,l/min$ if pipe L_2 is assumed to work correctly. Since at point C some other than the predicted value was observed, "$C = 4\,l/min$" is a symptom. At least one of the correctness assumptions that let to the prediction of "$C = 6\,l/min$" is wrong; i.e., the set $\{L_1, L_2\}$ forms a conflict.

In complex systems a single symptom may lead to a lot of conflicts. Note that the set of all components will always form a conflict, if at least one symptom has been observed. In order to reduce the complexity of the candidates generation step, it is useful to concentrate only on minimal conflicts. A conflict is called minimal, if no proper subset of it will form a conflict; any superset of a conflict always establishes a conflict.

3. *Candidate Generation.* Recall that a candidate is a set of components that can explain all observed symptoms, if all components in the set are broken. Clearly, in order to explain the observed symptoms, a candidate must contain at least one component of

each conflict. If we assume that $\{L_1, L_2\}$ was the only conflict in our example, then $\{L_1\}$ and $\{L_2\}$ would be the two minimal candidates.

The concept of Candidate Generation is able to cope with multiple faults. Let us consider that at point B the flow value $5\,l/min$ is measured. As a result $\{L_1\}$ and $\{L_2\}$ are the two minimal conflicts, and $\{L_1, L_2\}$ is the only (minimal) candidate.

Candidate sets establish hypotheses that indicate in which way the real system differs from the simulated model. The empty candidate set $\{\}$ is equivalent to the statement that all components work correctly.

4. *Candidate Discrimination.* A diagnosis process should produce a definite solution (diagnosis) of a diagnosis problem. Often there exist several minimal candidates, and additional knowledge in the form of additional observations is needed to differentiate between them. Thus candidate discrimination is realized by measuring point proposal for new observations. New observations can be obtained as follows:

   (a) Observations are made without modifying the system.

   (b) Observations are made after the modification of particular parameters of the system.

   Since the number of additional observations should be at a minimum, measuring points have to be chosen respecting their average gain of information. Note that a measuring point which provides the best discrimination between the actual candidates must not lead to a total minimum of measurements to identify a system's faults. Since the result of a measurement cannot be predicted, it is usually not possible to determine the measuring point that minimizes the total number of observations. De Kleer proposes the following equation to compute the gain of information [8]:

$$G(X) = \sum c_i \, ln \, c_i$$

   X denotes a measuring point, c_i denotes the number of minimal candidates that are consistent with the i-th possible measurement result. The sum is formed using all possible measurement results; the maximum average gain of information is provided by that point X for which $G(X)$ is at a minimum.

   In our example the only measurement point is B. Possible measurement results at B are $6\,l/min$ (if L_1 works correctly) and $4\,l/min$ (if L_2 works correctly). In either case the number of minimal candidates is 1; the computation of $G(B)$ is trivial: $G(B) = 1\,ln 1 + 1\,ln 1 = 0$.

5.3 Conflict Identification Based on the ATMS

The GDE explains deviations between the real system and the simulated model by retracting correctness assumptions that were made within in the model. Recall that if a predicted value v of the model does not match an observed value in the real system, the components involved in the inference of v form a *conflict*.

In order to identify such conflicts, a diagnosis system must be able to specify the origins of each inferred value. I.e., within each inference step the dependencies between the deduced value and its related correctness assumptions of the components have to be recorded. A mechanism which can do this dependency recording job is de Kleer's ATMS [6]. In the next subsection we will outline some of its concepts, details may be found in [6].

The Basic ATMS

An ATMS is a mechanism that records dependencies of the type "$\alpha \land \beta \rightarrow \gamma$". Note that an ATMS is not intended to realize the inference process of the problem solver. Rather it supports the problem solving process by maintaining well-chosen dependencies in the search space. In that sense the problem solver decides by itself which dependencies should be recorded at all by the ATMS; the ATMS is independent of the domain and the type of inference.

The ATMS constructs for each datum provided by the problem solver a node which consists of three parts: the datum, a label, and a set of justifications. The datum is treated as atomic and is stored independently of the problem solver's representation and semantics for it.

The label consists of a set of environments each of which represents a set of assumptions, that allow the datum to be inferred. Hence, the label shows the assumptions that must be made before the datum is known to be derivable. From the viewpoint of the GDE, all correctness assumptions regarding the hydraulic components form the assumption set of the ATMS.

Relations between assumptions and ATMS-nodes are established by the justifications. Each justification of a node consists of the antecedents of one of the problem solvers inferences used to infer the node's datum. Technically, a justification is represented as a propositional horn clause. Example: A behavior constraint that models the continuity condition of a pipe regarding some particular flow value is given with the following formula:

$$\neg\text{``}Q_{in} = 6\, l/min\text{''} \;\lor\; \neg\text{``}\texttt{pipe} = \texttt{ok}\text{''} \;\lor\; \text{``}Q_{out} = 6\, l/min\text{''}$$

A datum (or a node) is said to hold within an environment e, if the datum can be inferred from e and the justifications, using propositional logics. The job of the ATMS is the computation of a datum's environments. These computations are comprised of different set operations for the most part. If an inferred datum A is part of some justification supporting the datum B, A's assumptions are merged with those of B. The ATMS guarantees the minimality of an environment e, which means that the datum cannot be inferred from a proper subset of e.

Two important types of nodes should be mentioned: premises and nogoods. A premise depends upon no assumption and consequently, its label has an empty environment. A premise holds in every environment, it also has no justifications. Taking the standpoint of diagnosis, a premise corresponds to an observation made at the real system.

The problem solver can inform the ATMS that a datum, or a conjunction of data, is inconsistent. This can be thought of as an inference for the nogood datum, represented by

$\perp$. Hence, the label of $\perp$ is the set of those environments which are inconsistent. These environments are also called nogoods. Note that each superset of a nogood in turn establishes a nogood. Also note that the ATMS computes minimal nogoods.

The following properties of a label $l_n = \{e_1, \dots, e_k\}$ associated to a node n are essential with respect to the GDE[3]:

1. *Soundness.* The node n holds in every environment $e_i, e_i \in l_n$.

2. *Consistency.* $\perp$ cannot be inferred in some $e_i, e_i \in l_n$.

3. *Completeness.* Each consistent environment e from which n can be inferred is a superset of some $e_i, e_i \in l_n$.

4. *Minimality.* No $e_i \in l_n$ is a proper subset of some $e_j \in l_n$.

The GDE utilizes the label information of the ATMS to determine the necessary minimal conflicts: Let us assume that at some particular point of the hydraulic system a flow value of $Q_{10} = 5.5\, l/min$ has been predicted. I.e., in the ATMS there exists a node containing the datum "$Q_{10} = 5.5\, l/min$". The label of this node provides for the minimal sets of correctness assumptions of those components that are responsible for the inference of this datum. If now a flow value of $Q_{10} = 2.3\, l/min$ is observed at the real system, and the ATMS is informed about that fact, an appropriate premise will be established. Actually the ATMS has no idea of the hydraulic semantics that is associated with these two propositions, and thus the two nodes would live in the ATMS without causing a contradiction. The problem solver must explicitly establish a contradiction by means of the following justification:

$$\text{“}Q_{10} = 5.5\, l/min\text{”} \ \wedge \ \text{“}Q_{10} = 2.3\, l/min\text{”} \rightarrow \perp$$

Using this justification a contradiction can be inferred from all environments in the label of "$Q_{10} = 5.5\, l/min$", and the ATMS wil mark these environments as nogoods. Furthermore, the ATMS identifies all nodes that depend on "$Q_{10} = 5.5\, l/min$" and will mark them as nogoods as well. A key aspect is that these nogoods are minimal because each label's environments are minimal. I.e., the nogoods correspond to the minimal conflicts needed by the GDE.

Inexact Measurements

The identification of deviations between the real system and the simulated model, sometimes referred to as model-artifact differences, forms the base for our diagnosis process. When investigating a real system, errors in measurement cannot be excluded. We took this problem into consideration by introducing the special predicate `nearly-equal-p` when realizing the diagnosis system. Hence, the values computed by the inference system are mapped onto rounded values in the ATMS.

The use of this predicate involves problems in connection with multiple faults; usually a tolerance-relation is not transitive. The local decision whether a model-artifact difference

[3] These label properties are discussed in greater detail in [13].

results from an error in measurement or from a misbehaving component does not consider the propagation of errors.

Add to this, if a user decided to classify a model-artifact difference as a symptom, a revision would be impossible at a later stage: Symptoms are based on observations and are represented as premises within the ATMS. They cannot be redrawn.

5.4 A Closer Look at Candidate Generation

Within the candidate generation step the entire set of minimum candidates shall be computed. Remember that a candidate is a set of components that can explain all observed symptoms if all components in the candidate set are faulty. As a consequence a candidate will contain at least one element of each minimal conflict set.

Computing a minimal candidate is equivalent to the problem of computing a minimal hitting set, which is NP-complete [15]. Moreover, the number of minimal candidates can grow exponentially in the number of minimal conflicts: Consider a system that is built of $2n$ components; the observed symptoms shall result in n minimal conflicts, one minimal conflict for each pair of components $\{o_{2i}, o_{2i+1}\}$, $i \in 0 \ldots n-1$. Obviously the number of minimal candidates is 2^n.[4] If we restricted candidate generation to the detection of single faults[5], all minimal candidates could be computed in linear time [31].

The following subsections outline approaches for the generation (computation) of candidates.

Approximate Candidate Computation

Since the computation of a minimal candidate is NP-complete, one strategy to cope with the complexity is to give up the claim to an exact solution, i.e., to a candidate's minimum cardinality. Possibly an approximate solution is sufficient for a lot of real world hydraulic diagnosis problems. The considerations of this subsection are based on the assumption that the cost for the determination of an exact solution exceeds the cost for a redundant diagnosis, which is correct but not minimal.

The determination of a minimal candidate can be formulated as an optimization problem:

Problem: Minimal Candidate

Instance. A set $\mathcal{C}$ of minimal conflicts. Each element $C \in \mathcal{C}$ is a subset of a finite set of components O.

Configuration. A set $S \subseteq O$, where $S \bigcap C \neq \emptyset,\ C \in \mathcal{C}$.

[4] In the special case that all minimal conflict sets consist of two elements, the determination of a minimal candidate corresponds to the vertex cover problem of a "conflict graph" G. The components form the nodes of G, and each conflict set corresponds one-to-one to an edge of G.

[5] For complex hydraulic systems it is not useful to stick to the single fault assumption.

Solution. Any configuration.

Minimize. $c(S) := |S|$.

The problem Minimal Candidate establishes an optimization problem according to Lengauer [30]. c denotes the cost function of Minimal Candidate and maps an integer to each proper configuration.

The approximation towards an optimum solution to the above problem can be realized by selecting and modifying either an element from the space $\mathcal{S}$ of configurations or from the complement of $\mathcal{S}$. The idea of the latter is to efficiently compute a non-admissible solution[6] which is close to an optimum configuration. This approach is problematic— there is no proper measure to compute the quality of a candidate approximation that does not explain all observed symptoms. Thus we will concentrate on the former approach. The algorithm below computes an approximate solution for a minimum candidate.

Algorithm 5.1: Approximate Minimum Candidate

```
1. CANDIDATE $\leftarrow \emptyset$
2. $\mathcal{C} \leftarrow$ CONFLICTS
3. While $\mathcal{C} \neq \emptyset$
4.    Select a conflict $C := \{o_{c_1}, \ldots, o_{c_k}\}$ from $\mathcal{C}$
5.    CANDIDATE $\leftarrow$ CANDIDATE $\bigcup \{o_{c_1}, \ldots, o_{c_k}\}$
6.    Remove from $\mathcal{C}$ each conflict $C'$ with $C' \bigcap C \neq \emptyset$
7. Return CANDIDATE
```

Remarks. In line 1 and 2 the new candidate and the set of all conflicts are initialized. Within the lines 3 to 5 a hitting set over the minimum conflicts is computed, which is stored into CANDIDATE. The algorithm terminates because at least one element is removed from $\mathcal{C}$ within each pass of the While-loop. Note that if the conflict sets are sorted, all set operations can be done in $O(n)$, where n denotes the total number of components.

Aside from its time complexity the practical advantage of algorithm 5.1 depends on the deviation between an approximate solution, computed by algorithm 5.1, and the optimum. According to Lengauer [30] we define a metric by which the error of some algorithm A can be measured:

$$\text{ERROR}(A) := \max_{p \in I} \frac{c(A(p))}{c(\text{opt}(p))}$$

I denotes the set of all problem instances, $A(p)$ is the solution computed by algorithm A, and opt(p) is the optimum solution for p.

Lemma. The error of algorithm 5.1 is $\max\{|C| \mid C \in \text{CONFLICTS}\}$.

Proof. Let $\mathcal{C}$ be all minimal conflicts, and let $\mathcal{C}' \subseteq \mathcal{C}$ be those minimal conflicts that are actually selected within line 4. Because of the Remove-operation in line 6, no conflicts in $\mathcal{C}'$ have an element in common. Hence, the following relations hold:

$$|\text{CANDIDATE}| \;=\; \sum_{C \in \mathcal{C}'} |C| \;\leq\; |\mathcal{C}'| \cdot \max\{|C| \mid C \in \mathcal{C}\} \tag{5.1}$$

[6] Each set of components which does not cover all conflict sets represents a non-admissible solution.

To cover all conflicts, a (minimum) candidate must contain at least one element from each conflict set in $\mathcal{C}$ and $\mathcal{C}'$ respectively. Since the conflict sets in $\mathcal{C}'$ establish mutually exclusive sets, no element of a minimum candidate is contained in more than one conflict set of $\mathcal{C}'$. Hence,

$$|\mathcal{C}'| \leq |\text{CANDIDATE}^*| \tag{5.2}$$

From (5.1) follows

$$\frac{|\text{CANDIDATE}|}{\max\{|C| \mid C \in \mathcal{C}\}} \leq |\mathcal{C}'| \tag{5.3}$$

Putting together (5.2) and (5.3) results in

$$\frac{|\text{CANDIDATE}|}{\max\{|C| \mid C \in \mathcal{C}\}} \leq |\text{CANDIDATE}^*| \Leftrightarrow \frac{|\text{CANDIDATE}|}{|\text{CANDIDATE}^*|} \leq \max\{|C| \mid C \in \mathcal{C}\}$$

◇

The error of algorithm 5.1 is bound by the maximum cardinality of the minimum conflict sets. I.e., if one minimal conflict contains n elements, the computed candidate may contain n of times as much as elements as a minimal candidate. This result is acceptable only in the special case that all conflict sets contain two elements. Thus, in the next subsection, we will present a procedure of de Kleer and Williams [10], which incrementally computes the minimal candidates.

Candidate Computation due to de Kleer and Williams

The algorithm of de Kleer and Williams copes with multiple faults and works incrementally. It is outlined below.

Algorithm 5.2: Incremental Diagnosis of Multiple Faults

```
1.   CANDIDATES ← $\emptyset$
2.   Propose observation
3.   Generate CONFLICTS

4.1  $\forall D, D \in$ CANDIDATES
4.2    $\forall C, C \in$ CONFLICTS
4.2      If $D \bigcap C = \emptyset$ Then
4.3        CANDIDATES := CANDIDATES $\setminus D$
4.4        $\forall o, o \in C$ : CANDIDATES := CANDIDATES $\bigcup \{D \bigcup \{o\}\}$
4.5      Remove subsumed candidate sets in CANDIDATES[a]

5.   Stop, if CANDIDATES is sufficiently detailed.
6.   Goto step 2.
```

[a] Let C_1 and C_2 be two sets. C_1 is subsumed under C_2 if $C_2 \subseteq C_1$.

How does the candidates computation work? When diagnosis starts, CANDIDATES is the empty set. With each new observation the symptom propagation updates the sets of minimal conflicts. The steps under 4 realize the incremental computation of the candidates. The following example illustrates the algorithm.

Let us assume that in the course of the diagnosis of a hydraulic system the two minimal conflicts $\{A, B, C\}$ and $\{C, B\}$ have been generated. If these sets are the only conflicts, the candidate generation tree looks like depicted in figure 5.4.

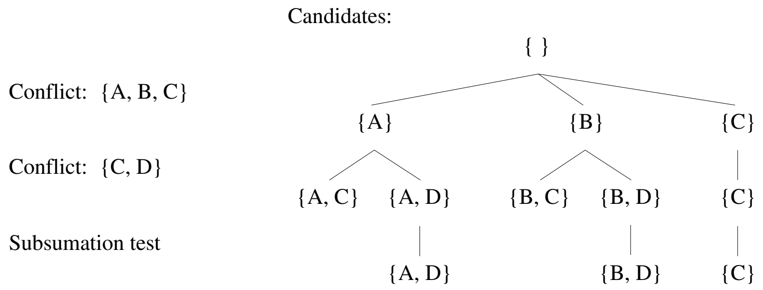


Figure 5.4: *Incremental candidate generation according to de Kleer and Williams;* $\{A, B, C\}$ *and* $\{C, D\}$ *are the only conflicts.*

If, in a further course of the diagnosis process, a new observation has been made, which let to new sets of minimal conflicts, $\{A, B\}$ and $\{D, E\}$, the candidate generation tree is updated as shown in figure 5.5.

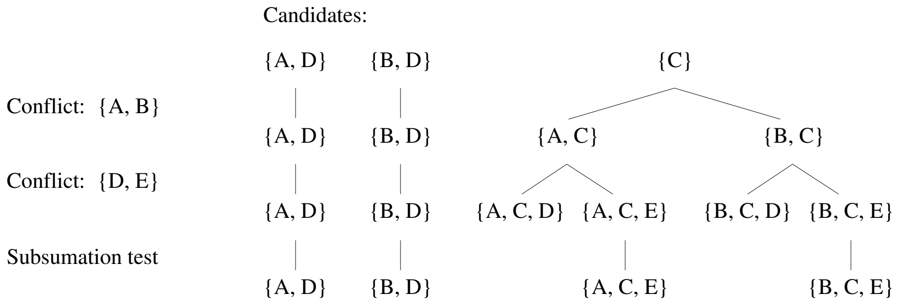


Figure 5.5: *Continuation of the above candidate generation example;* $\{A, B\}$ *and* $\{D, E\}$ *are the only conflicts now.*

The example shows how algorithm 5.2 works; moreover, it discloses another interesting effect: Obviously the computed candidate sets $\{A, C, E\}$ and $\{B, C, E\}$ are not minimal, the component C is not contained in any minimal conflict set. This rises the question if the candidate generation algorithm of de Kleer and Williams works correctly.

Since we employed this type of candidate generation scheme within our prototypic diagnosis system, it was necessary to investigate its correctness. The original paper of de Kleer and Williams does not provide for a formal proof of the correctness of algorithm 5.2. Leveling presents an investigation of related issues and shows the correctness of the algorithm [31]. In this place we will not further elaborate on this question.

Complexity Considerations

As shown before, the time complexity of a diagnosis problem considering multiple faults grows exponentially in the number of a system's components. Consequently the total number of processed components in a hydraulic should be kept as small as possible. This objective can be achieved by means of the following concepts:

1. Comprising components of a hydraulic system by the introduction of substitute resistances.
2. Focusing on particular parts of a hydraulic system during the diagnosis process.

In the following we consider a hydraulic system as a network consisting of hydraulic resistances, sources, and sinks. A certain class of networks, the so-called series-parallel networks, can be represented by a hierarchy of substitute resistances. Loosely speaking, series-parallel networks are composed of series and parallel connections of resistances only. Series-parallel networks can be mapped onto series-parallel graphs, which are defined in [1] or in [30].

The decomposition levels of a series-parallel network define a tree whose leafs represent the components of the hydraulic system. Figure 5.6 shows an example.

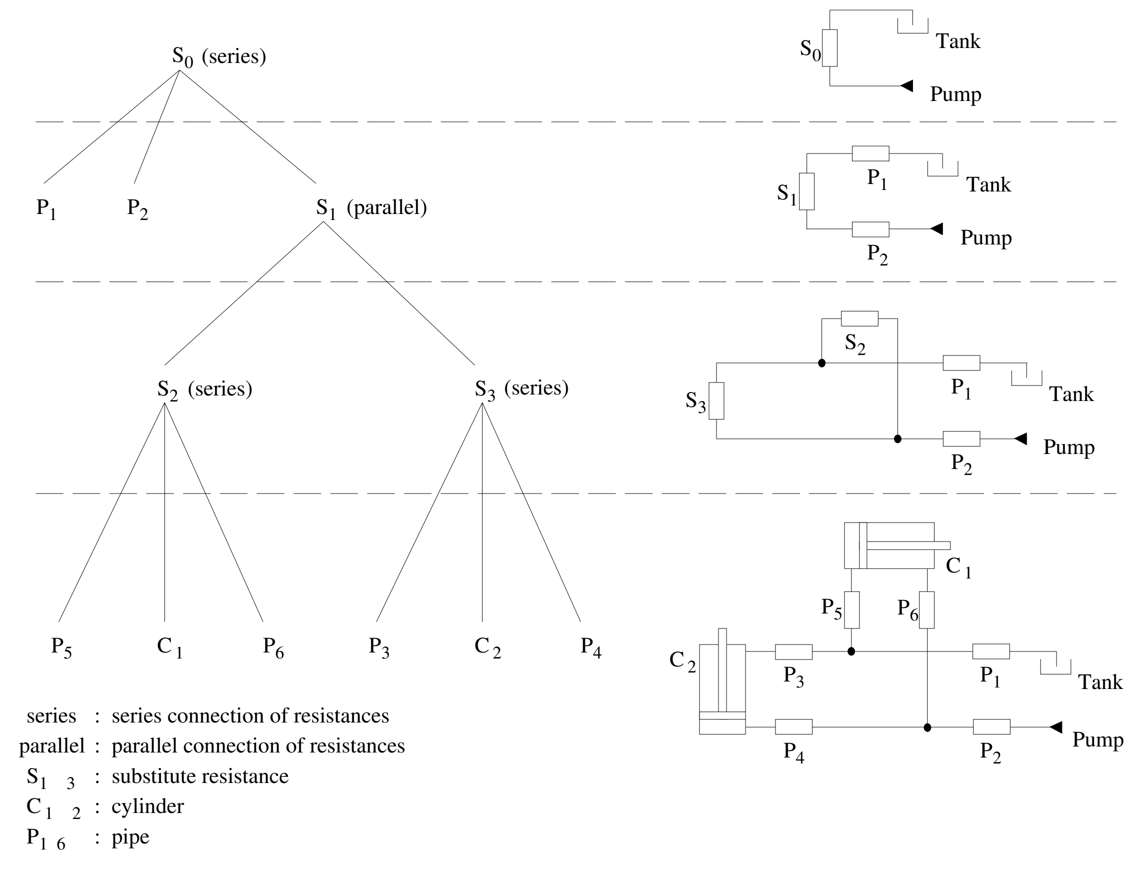


Figure 5.6: *Hydraulic system represented at different levels of decomposition.*

The inner nodes of the decomposition tree denote the substitute resistances S. A substitute resistance comprises its descendant nodes, which are connected either in series or in parallel. In this way the entire network can be represented through one substitute resistance, the root S_0. Note that the nodes on each direct path from the root to a leaf comprise series and parallel connections in an alternating sequence. The decomposition tree of a series-parallel network can be computed in linear time [30].

Given such a decomposition tree, the strategy for focussing works at follows. If a symptom has been observed at component o, then all nodes of the direct path between the root and o are expanded. Expanding a substitute resistance S means to employ all direct descendants of S to represent the hydraulic system. When diagnosis starts the entire system is represented by the substitute resistance associated with the root. If, for example, a symptom has been observed at cylinder C_1, the nodes S_0, S_1, and S_2 will be expanded, and the hydraulic system consists of C_1, P_1, P_2, P_5, P_6, and S_3.

5.5 A Constraint System for Diagnosis

Within our diagnosis approach a hydraulic plant is modeled using constraints. As described earlier in chapter 2 a constraint consists of both a set of variables and a relation defined upon these variables. Constraint processing will realize the inference of component behavior, which is a constructive task, as well as the test of inconsistencies relating the observed symptoms, which is some kind of destructive task.

Concepts of the Constraint System

From a diagnostic point of view, hydraulic components need not to be modeled at a very detailed physical level. Simplified equations, which qualitatively describe component behavior, are in most cases sufficient for a global diagnosis process at the circuit level. Figure 5.7 shows a black-box representation of a simplified cylinder.

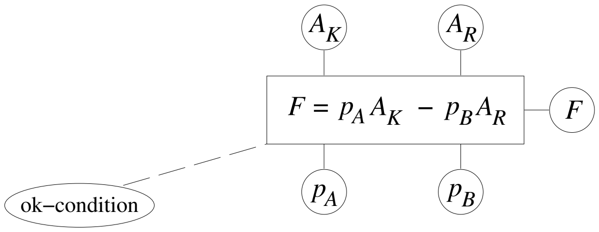


Figure 5.7: *Graphical description of a cylinder constraint*

Note that the correctness assumption (OK-condition) establishes an obligatory part of each behavior constraint: It determines whether a constraint can be applied at all.

We realized our diagnosis prototype using the ATCON-language to formulate the component constraints [13]. In the ATCON system to each constraint-variable a structure is attached with slots where, aside from the variable's value, information regarding both constraint dependencies and parameter dependencies is stored. ATCON is based on an ATMS; each constraint-variable also establishes an ATMS node.

A constraint in ATCON enforces the fulfillment of its associated relation by means of rules. These rules are defined for each variable. A rule is of the following form:

(⟨`head`⟩ ⟨`body`⟩ ⟨`computation rule`⟩)

⟨`head`⟩ contains the name of the variable to be computed; ⟨`body`⟩ contains a subset of a constraint's variables each of which must be known before the rule can be applied; ⟨`computation rule`⟩ provides a LISP function that computes the value for the variable in ⟨`head`⟩, dependent on the values supplied in ⟨`body`⟩.

The subsequent constraint example shows one rule of the definition of the balance of forces in a cylinder. It defines the dependencies of the variable p_A from the other variables involved:

```
(p_A (p_B F A_K A_R OK) (if (or (not OK)
                                (nearly-zero-p A_K))
                            :DISMISS
                            (/ (+ F (* p_B A_R)) A_K)))
```

The symbol `:DISMISS` denotes a particular keyword which informs the inference machine that no value can be computed by this rule if the `or`-clause evaluates to *True*.

Symptom Propagation in the Constraint Network

Together all components' constraints form a constraint network. Each two components that are mechanically connected in the hydraulic system share at least one variable in the constraint network. A cylinder with a pipe for instance have one flow variable and one pressure variable in common. Hence, values for variables can be propagated in the network.

In ATCON value propagation works as follows. If a constraint provides sufficient information to deduce a new value for some variable, a particular rule can fire. For each rule that fires, ATCON creates a new node within the underlying ATMS. The computed value for the rule's ⟨`head`⟩-variable becomes the datum of this node. The values for the variables in the ⟨`body`⟩-part of the rule become the antecedent nodes of the justification. Finally, the new node's label is computed by the ATMS as described in a former subsection.

Note that also for the components' correctness assumptions (OK-conditions) ATMS-nodes[7] exist. Each justification that is derived from a constraint thus contains the correctness assumption of the component which is associated to this constraint. As a result correctness assumptions occur in the label of every propagated value v; they specify the set of correctly working components responsible for v.

If because of a new symptom observation a variable of the constraint system is modified, the involved constraints are tickled and checked whether their rules can fire. This symptom propagation works according to the following rule:

[7] These ATMS-nodes are of the type "assumption".

Algorithm 5.3: Symptom Propagation

```
While a new value has been inferred
    Select some variable $v$ that has been modified.
    For each constraint $c$ that is defined upon $v$:
        Apply all rules of $c$ that can fire.
```

Remarks. The above algorithm processes the rules in a forward chaining manner. In order to avoid cycles during propagation, the processed constraints will be marked. Note that constraint propagation does not necessarily follow the physical flow of energy in the hydraulic system.

Modeling Layers

Three modeling layers of a hydraulic plant can be identified in the diagnosis system (cf. figure 5.8). The topmost layer contains the representation of the hydraulic plant's topology; it is comprised of components, connections, and pipes. This layer corresponds to the user's view to a hydraulic plant and is created while drawing a circuit with $^{\mathrm{art}}$*deco*.

The second layer, which is called constraint layer here, realizes the component behavior from the standpoint of diagnosis. The functional models of the components are exploited to create both the behavior constraints of ATCON and the ATMS nodes.

Taking the observed symptoms as input, the second layer constructs via constraint propagation the lowest layer, the justification layer.

Note that a user is faced only with the topmost layer.

Discussion

Constraint propagation is one possibility to realize behavior prediction for technical systems. The description and the processing of constraints as outlined in this section has been used in different diagnosis systems, especially in connection with the GDE [13]. Bound up with such an approach are the following advantages:

- The constraint network can be easily constructed.
- Continuous connections can be modeled.
- The computation of local constraints is efficient.
- The formulation of local dependencies is very intuitive and simplifies the process of knowledge acquisition.

A major drawback of the above constraint propagation approach is that it can fail in computing all unknown variables, although a globally consistent solution exists. Recall that the identification of conflicts is realized by constraint propagation in cooperation with the ATMS. An imperfect determination of variables in the constraint network may result in

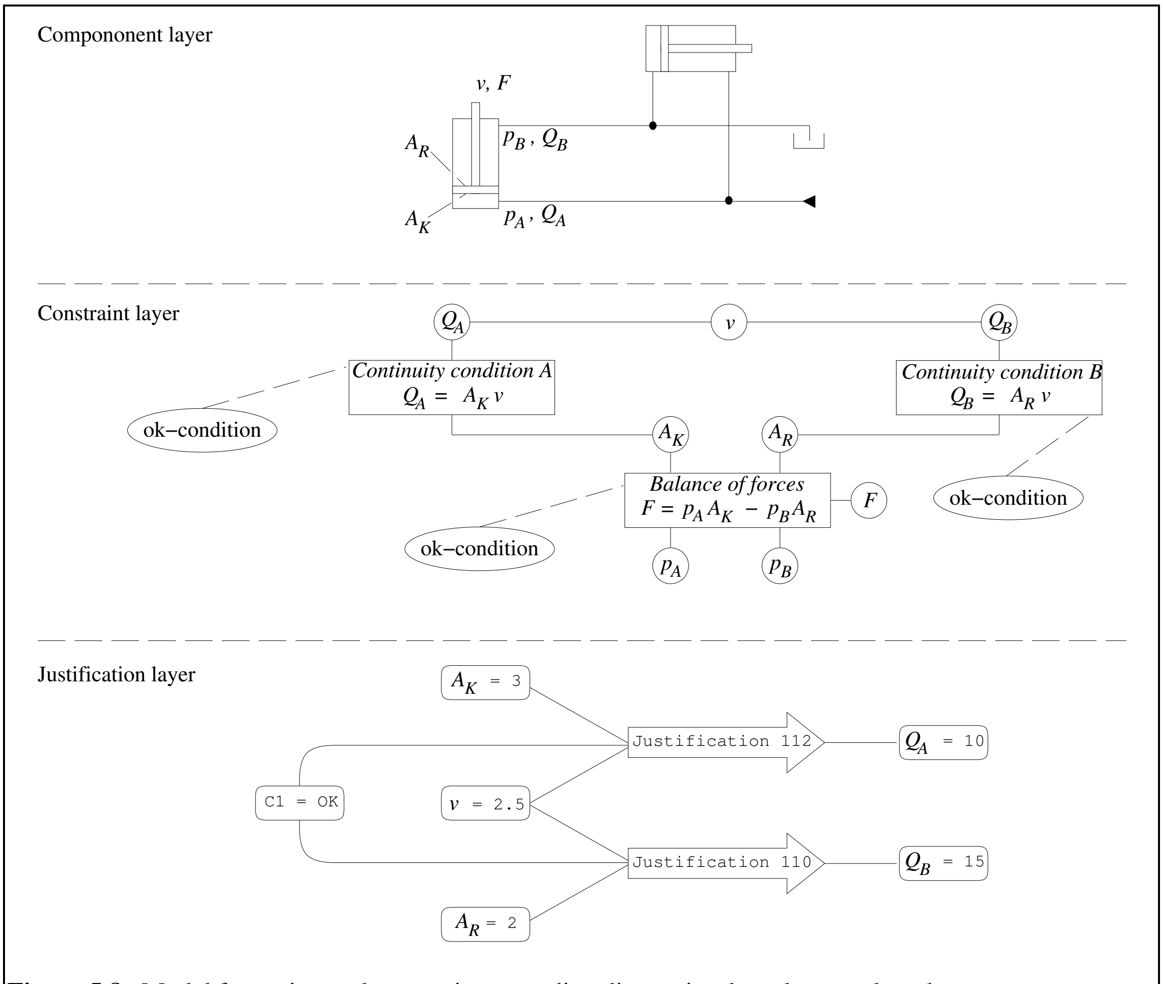


Figure 5.8: *Model formation and processing regarding diagnosis takes place on three layers*

smaller conflict sets. Hence the GDE will not be able to propose the most discriminating measuring point; as a consequence, the number of observations will not be minimum.

References

[1] H. Booth and R. Tarjan. Finding the Minimum-Cost Maximum Flow in a Series-Parallel Network. *Journal of Algorithms*, 15:416–446, 1993.

[2] D. Brown and B. Chandrasekaran. *Design Problem Solving*. Morgan Kaufmann Publishers, 1989.

[3] T. Bylander and B. Chandrasekaran. Generic Tasks for Knowledge-based Reasoning: the “right” Level of Abstraction for Knowledge Acquisition. *Int. J. Man-Machine Studies*, 26:231–243, 1987.

[4] B. Chandrasekaran and R. Milne. Reasoning About Structure, Behavior, and Function. *SIGART Newsletter*, Juli 85(93):4–59, 1985.

[5] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32:281–331, 1987.

[6] J. de Kleer. An Assumption-based Truth Maintenance System. *Artificial Intelligence*, 28:127–162, 1986.

[7] J. de Kleer. Problem Solving with the ATMS. *Artificial Intelligence*, 28:197–224, 1986.

[8] J. de Kleer. Using Crude Probability Estimates to Guide Diagnosis. *Artificial Intelligence*, 45:381–391, 1990.

[9] J. de Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24:7–83, 1984.

[10] J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97–130, 1987.

[11] R. Dechter and J. Pearl. Network-based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, pages 1–38, 1988.

[12] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[13] K. D. Forbus and J. de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, 1993.

[14] E. C. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11):958–966, Nov. 1978.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[16] J. Gosling. *Algebraic Constraints*. Dissertation, Carnegie-Mellon University, Department of Computer Science, May 1983.

[17] H.-W. Güsgen. CONSAT—*A System for Constraint Satisfaction*. Dissertation, Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, Nov. 1987.

[18] M. Hoffmann. Algorithmen zur Verarbeitung von topologischen Informationen in Netzwerken. Diploma thesis, Gerhard-Mercator-Universität - GH Duisburg, Fachbereich Mathematik / Informatik, Sept. 1993.

[19] U. Husemeyer. Entwurf und Realisierung einer Verhaltensbeschreibungssprache für technische Expertensysteme. Diploma thesis, University of Paderborn, Department of Mathematics and Computer Science, 1995.

[20] IntelliCorp. KEE *User's Guide*. IntelliCorp, Inc, 1975 El Camino Real West, California, 1988.

[21] A. Kecskeméthy. MOBILE—An Objectoriented Tool-Set for the Efficient Modeling of Mechatronic Systems. In M. Hiller and B. Fink, editors, *Second Conference on Mechatronics and Robotics*. IMECH, Institut für Mechatronic, Moers, IMECH, 1993.

[22] J. Kippe. Komponentenorientierte Repräsentation technischer Systeme. In H. W. Früchtenicht, editor, *Technische Expertensysteme: Wissensrepräsentation und Schlußfolgerungsverfahren*. R. Oldenbourg Verlag, München, Wien, 1988.

[23] H. Kleine Büning and B. Stein. Supporting the Configuration of Technical Systems. In M. Hiller and B. Fink, editors, *Second Conference on Mechatronics and Robotics*. IMECH, Institut für Mechatronic, Moers, IMECH, 1993.

[24] B. Kuipers. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence*, 24:169–203, 1984.

[25] R. Lemmen. Akquisition und Analyse von Wissen zur Inbetriebnahme von hydraulischen translatorischen Anlagen. Diploma thesis, Gerhard-Mercator-Universität - GH Duisburg, Inst. f. Meß-, Steuer-, und Regelungstechnik, 1991.

[26] R. Lemmen. Modellbasierte Prüfung hydraulischer Anlagen. Technical Report MSRT7/92, Gerhard-Mercator-Universität - GH Duisburg, 1992.

[27] R. Lemmen. Entwurf und Implementierung eines wissensbasierten Verfahrens für die Inbetriebnahme technischer Anlagen am Beispiel translatorischer hydraulischer Antriebe. Technical Report MSRT 5/95, Gerhard-Mercator-Universität - GH Duisburg, 1995.

[28] R. Lemmen. *Zur automatisierten Modellerstellung, Konfigurationsprüfung und Diagnose hydraulischer Anlagen mit dem Beispiel tankdruckangehobener Differentialzylinderantriebe.* Number 503 in Fortschrittsberichte VDI, Reihe 8: Mess-, Steuer- und Regelungstechnik. VDI-Verlag, DÃ$\frac{1}{4}$sseldorf, 1995.

[29] R. Lemmen and B. Stein. Wissensbasierte Konfigurationsprüfung hydraulischer Anlagen. *at – Automatisierungstechnik*, 3, 1994.

[30] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* B. G. Teubner, Stuttgart, 1990.

[31] U. Leweling. Möglichkeiten und Grenzen der modellbasierten Diagnose bei hydraulischen Anlagen. Diploma thesis, Universität-GH Paderborn, FB 17 Mathematik / Informatik, 1995.

[32] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 24:169–203, 1994.

[33] Math Works Inc. SIMULINK *User's Guide.* Math Works Inc., Nattik, Massachusetts, 1992.

[34] Y. Nakashima and T. Baba. OHCS: Hydraulic Circuit Design Assistant. In *First Annual Conference on Innovative Applications of Artificial Intelligence*, pages 225–236, Stanford, 1989.

[35] J. K. Ousterhout. TCL: An Embeddable Command Language. In *USENIX Conference*, 1990.

[36] M. Piechnick and A. Feuser. MOSIHS – Programmsystem zur Simulation komplexer elektrohydraulischer Systeme. In *AFK, Aachener Fluidtechnisches Kolloquium.* Mannesmann Rexroth GmbH, Lohr, Germany, 1994.

[37] F. Puppe. *Einführung in Expertensysteme.* Springer-Verlag, 1988.

[38] F. Puppe. *Problemlösungsmethoden für Expertensysteme.* Springer-Verlag, 1990.

[39] H. R. Schwarz. *Numerische Mathematik.* B. G. Teubner, Stuttgart, 1986.

[40] B. Stein. *Functional Models in Configuration Systems.* Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1995.

[41] B. Stein and R. Lemmen. ARTDECO: A System which Assists the Checking of Hydraulic Circuits. In *Workshop for Model-based Reasoning, ECAI '92*, 1992.

[42] P. Struß. Assumption-based Reasoning about Device Models. In H. Früchtenicht, editor, *Wissensrepräsentation und Schlußfolgerungsverfahren.* Oldenbourg Verlag, München, 1988.

[43] M. Suermann. Wissensbasierte Modellbildung und Simulation von hydraulischen Schaltkreisen. Diploma thesis, Universität-GH Paderborn, FB 17 Mathematik / Informatik, 1994.

[44] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2):146–160, June 1972.

[45] H. VoÃ§. *Representing and Analyzing Causal, Temporal, and Relations of Devices*. Dissertation, Universität Kaiserslautern, Fachbereich Informatik, 1986.

[46] J. Weiner. *Aspekte der Konfigurierung technischer Anlagen*. Dissertation, Gerhard-Mercator-Universität - GH Duisburg, FB 11 Mathematik / Informatik, 1991.