Process-oriented Consistency-based Diagnosis - Theory, Implementation and Applications

Ulrich Heller
The existing theory of consistency-based diagnosis and its implementations have proven successful in a number of technical applications. However, they turn out to be inherently limited to a very specific class of systems to be diagnosed: They are tailored for artifacts consisting of components in a fixed structure and they are aimed at a particular kind of diagnosis and repair, namely failing components and their replacement.

This thesis presents a generalization of the theory of consistency-based diagnosis, applicable to a much wider class of systems and diagnosis problems, while preserving the approach based on "first principles" and the core of the existing logical foundations, i.e. the revision of an initial system model until consistency with a specified criterion is achieved.

As a first contribution, a logical reconstruction of a process-oriented modeling paradigm is presented, which is shown to be a generalization of the component-oriented paradigm. This is the foundation for the specification, as well as the automated composition and revision of system models to be used in checking consistency.

Compared to previous approaches, that define a single diagnostic task, the presented approach differentiates between situation assessment, i.e. revising an initial model to establish consistency with given observations, and therapy recognition, i.e. revision in order to establish consistency with a specified goal. As a second contribution, the concise characterization of solutions to these tasks in a non-monotonic logic formalism is presented.

The third contribution is an algorithmic approach to computing the specified solutions, by transforming the non-monotonic theory into monotonic composition plus constraint-based consistency checking, potentially followed by a search for structural revisions. These algorithms form the basis of an implemented prototype, the Generalized Diagnosis Engine (GDE).

Several extended application examples, concerned with water treatment, medical theory validation, and electrical circuits, highlight different aspects and strengths of the approach and provide an introduction to problem-solving with the Generalized Diagnosis Engine.
Acknowledgments

I would like to thank Prof. Peter Struss for his supervision and support of my work on this thesis, but in particular for pointing out this very interesting direction of research to me in the first place, for many intense discussions on the subject, and for his unique perspective, unfalteringly focused on the generalization of an issue under consideration.

Thanks to Prof. Wilfried Brauer for agreeing to review this thesis, which profited notably from his suggestion to research additional applications and examples, and to Prof. Zenger for agreeing to serve on my committee.

Special thanks to the members of the Model-based Systems and Qualitative Reasoning Group Munich (MQM), especially Andreas Malik and Martin Sachenbacher for many interesting discussions and insightful comments and suggestions contributing to the theory in various stages of its development (including the thorough proofreading of the final draft by Andreas). I will miss the continuous stream of their new ideas, hypotheses and theories as well as the amicable, but tough challenges presented to my own ones.

Thanks to Robert Illner, Christopher Schmid, and Andrea Brkic for their early implementations of qualitative influence diagrams, model transformations and process-oriented model composition, all of which represented important test-beds for the refinement of the theory and the development of the current prototype.

Also, the participants of the 9th International Workshop on Principles of Diagnosis (DX-98) in Cape Cod have commented on an early draft of this work and supplied helpful related literature.

I'll mention Raymond Reiter, Johan de Kleer, and Kenneth Forbus as representatives for many other researchers who have assembled the solid foundations, I had the opportunity to build upon.

The Universidade Federal do Rio Grande do Sul (UFRGS) as well as the Departamento Municipal de Agua e Esgotos (DMAE) of Porto Alegre in Brazil have supported this work in active collaboration on the subject of diagnosis and situation assessment for their water treatment plants and the ecosystems under their surveillance. The German Ministry of Education and Research (BMBF) and the Brazilian Conselho Nacional de Pesquisa (CNPq) have funded this exchange in the framework of their bilateral scientific-technical cooperation (STC). Special thanks go to Prof. Waldir Roque from the Department of Applied Mathematics at UFRGS and Dr. Elenara Correa Lersch from the research department of DMAE for their support of the project and for giving me the opportunity to be challenged by real problems demanding a solution.

Last but not least, I extend my gratitude to Barbara, who deserves an academic degree of her own for researching creative ways of supporting me and bearing with me through these recent years of intense work.
Table of Contents

1 Introduction ................................................................. 1
2 A Motivating Example .................................................. 5
3 The Standard Approach to Consistency-based Diagnosis .......... 9
  3.1 The Principle of Consistency-based Diagnosis ......................... 9
  3.2 Computing Diagnostic Candidates ........................................ 11
  3.3 Revisiting the Example ................................................... 12
4 Foundations of the Generalized Theory of Consistency-based Diagnosis ........................................ 15
5 The Modeling Approach ................................................... 19
  5.1 Process-oriented Modeling .............................................. 19
  5.2 Generalizing Component-oriented Modeling ......................... 21
  5.3 Formally: The Modeling Approach ...................................... 21
    5.3.1 Ontology .................................................................. 23
    5.3.1.1 Object Types ....................................................... 23
    5.3.1.2 Relations .......................................................... 25
    5.3.2 Quantities .................................................................. 27
    5.3.3 Behavior Constituent Types .......................................... 28
      5.3.3.1 Structural Conditions ........................................... 29
      5.3.3.2 Quantity Conditions ............................................. 31
      5.3.3.3 The Fundamental Laws of Behavior Constituents .............. 32
      5.3.3.4 Structural Effects ................................................ 33
      5.3.3.5 Quantity Effects .................................................. 35
      5.3.3.6 Influence Resolution ............................................. 37
    5.3.4 Basic Axioms ............................................................ 39
    5.3.5 Closure of the Domain Theory ....................................... 39
    5.3.6 Situation Description ................................................ 41
6 Characterization of Tasks and Solutions ................................ 43
  6.1 Differentiation of Diagnostic Tasks ..................................... 44
    6.1.1 Situation Assessment ................................................. 44
    6.1.2 Therapy Recognition ................................................ 45
    6.1.3 Specification of Revisables .......................................... 45
  6.2 Formally: Solutions ....................................................... 46
    6.2.1 Completion of the Situation Description ........................... 46
    6.2.2 Minimality of Completions ......................................... 48
    6.2.3 Completion and Retraction .......................................... 49
  6.3 Candidate Ranking ......................................................... 50
7 The Generalized Diagnosis Engine (G*DE) ........................... 51
  7.1 Key Idea of the Approach ................................................. 51
    7.1.1 Computing Completions - The Extended Forward Space .......... 51
    7.1.2 Formalization of the Approach ...................................... 54
      7.1.2.1 Basic Definitions - Potential Elements and Behavior Constituents ......... 54
      7.1.2.2 Definition of the Extended Forward Space - Instantiation Rules ............... 55
      7.1.2.3 Reasoning Within the Extended Forward Space - Activation Rules .......... 57
List of Figures

Figure 2.1: Example scenario - observation of metallic taste ................................................................. 5
Figure 2.2: Example - situation assessment ............................................................................................ 6
Figure 2.3: Example - possible countermeasures ("therapy recognition") .................................................. 6

Figure 4.1: Prediction and consistency check for situation assessment (top) and therapy recognition (bottom) 15
Figure 4.2: The basic cycle of a consistency-based problem solver .......................................................... 16
Figure 4.3: Bridging the gap by automated model composition .................................................................. 16
Figure 4.4: Problem solving based on model revision .............................................................................. 17

Figure 5.1: Overview of the Modeling Sections ........................................................................................ 22
Figure 5.2: Specificity of the Modeling Sections ...................................................................................... 22
Figure 5.3: User defined parts .................................................................................................................. 23
Figure 5.4: Revisable parts ....................................................................................................................... 23
Figure 5.5: Example of object type hierarchy ............................................................................................ 24
Figure 5.6: Example of relation definitions .............................................................................................. 26
Figure 5.7: Example of quantity associations .......................................................................................... 28
Figure 5.8: Example of behavior constituent type definition: IronRedissolving ........................................ 28
Figure 5.9: Structural conditions of IronRedissolving .............................................................................. 29
Figure 5.10: Structural conditions of a Pipe component model .............................................................. 30
Figure 5.11: Quantity conditions of IronRedissolving .............................................................................. 32
Figure 5.12: Structural effects of IronRedissolving .................................................................................. 34
Figure 5.13: Quantity effects of IronRedissolving ................................................................................... 36
Figure 5.14: Pre-defined influence types .................................................................................................. 38

Figure 7.1: Completion of situation description: "forward" vs. "backward" and extensions of predicates .......... 52
Figure 7.2: Multiple completions and the interaction between completions and restrictions ...................... 52
Figure 7.3: The "extended forward space" as a superset of the forward completion ...................................... 53
Figure 7.4: General consistency-based problem solving (cf. figure 4.4) .................................................... 60
Figure 7.5: An instance of the IronRedissolving behavior constituent from section 5.3.3 ............................. 62
Figure 7.6: Guard variables and constraints for the structural conditions .................................................. 63
Figure 7.7: Guard variables and constraints for encoding the quantity conditions ...................................... 64
Figure 7.8: Guard variables and "implication influences" for encoding the structural effects ...................... 65
Figure 7.9: Guard variables and constraints for encoding the quantity effects .......................................... 67
Figure 7.10: The complete influence diagram fragment for the behavior constituent instance .................... 67
Figure 7.11: Architecture of G^DE (abstract view) .................................................................................. 82
Figure 7.12: Architecture of G^DE (detailed view) ................................................................................... 83
Figure 7.13: Screenshot of Visio interface (editing a behavior constituent) .................................................. 84

Figure 8.1: Object type hierarchy for the water treatment examples ........................................................... 88
Figure 8.2: Relation definitions for the water treatment examples ............................................................ 89
Figure 8.3: Quantity associations for the water treatment examples .......................................................... 89
Figure 8.4: Behavior constituent type "Iron Redissolving" ......................................................................... 89
Figure 8.5: Behavior constituent type "Iron Ascending" ............................................................................ 90
Figure 8.6: Behavior constituent type "Iron Transport" ............................................................................. 90
Figure 8.7: Behavior constituent type "Iron Perception" ............................................................................ 91
Figure 8.8: Behavior constituent types "Alkalization" and "Oxidation" ......................................................... 91
Figure 8.9: Informal description of the scenario (compare figure 2.1 from section 2) ................................. 91
Figure 8.10: Formal encoding of the scenario description for the "sedimental iron" example ...................... 92
Figure 8.11: A simple solution to the "metallic taste" scenario ................................................................. 93
Figure 8.12: The composed model for the intermediate scenario (including all created structural elements and behavior constituent instances) .............................................. 94
1 Introduction

Model-based diagnosis systems have been developed and applied over more than 15 years, and, meanwhile, they have gained maturity and reached industrial applications. We mention only a few representative projects focusing on the diagnosis of power transmission networks ([Beschta et al. 1993]), power generation turbines ([Milne et al. 1994]), autonomous space crafts ([Williams/Nayak 1996]), and car subsystems ([Bidian et al. 1999], [Sachenbacher et al. 2000]), and the German joint project "Intelligent Diagnosis in Industrial Applications" ([Hotz et al. 2000]). Model-based diagnosis systems represent a remarkable success, which has been achieved on the basis of a rigorous theoretical foundation.

On the other hand, it is the step towards practical applications that raises the question about the scope of applicability. Can we expect to find, theoretically and practically, solutions to all of the various diagnostic problems by applying the existing "theory of diagnosis based on first principles" ([Reiter 1987])? Do we have the "General Diagnosis Engine" ([de Kleer/Williams 1987])?

The foundation of consistency-based diagnosis is the search for behavior models, \( \text{MODEL} \), of a system to be diagnosed that are consistent with the given observations, \( \text{OBS} \), of the system's behavior,

\[
\text{MODEL} \cup \text{OBS} \not\models \perp
\]

and, hence, are candidates for representing the actual malfunctioning system. More precisely, such a candidate is characterized by assigning a behavior mode, \( \text{mode}_i \), to each component, \( C_i \), of the system such that this assignment is consistent with the system description, \( \text{SD} \), (comprising behavior descriptions of the various modes and the system structure) and the observations:

\[
\text{SD} \cup \{ \text{mode}(C_i) \mid C_i \in \text{COMPS} \} \cup \text{OBS} \not\models \perp.
\]

Modes correspond to the correct (intended) behavior of components and their faults. Consistent mode assignments are generated as incremental (minimal) revisions of an initial model that assigns correct behavior to all components. In the theory, the system description is required to be a set of first order sentences ([Reiter 1987], [de Kleer et al. 1992]), and in most implementations, it is a set of equations or relations (constraints) ([de Kleer/Williams 1987], [de Kleer/Williams 1989], [Struss/Dressler 1989]). It appears that a theory of diagnosis can't possibly be more general.

Actually, we argue that, on the one hand, the theory is far too general. If it is meant to exploit first principles for diagnostic purposes, this would have to be reflected in a special structure and content of the system description, \( \text{SD} \), instead of regarding it as some arbitrary set of first order sentences, which does, for instance, not exclude statements that are empirical associations and heuristics instead of physical laws. Hence, a theory of diagnosis based on first principles has to be more specific about \( \text{SD} \).

On the other hand, the theory is far too specific w. r. t. to the kind of systems and diagnostic tasks it covers. A closer analysis reveals that the theory and even more the implemented systems are tailored

- to diagnose faults due to broken components only,
- in properly designed artifacts,
- comprising a fixed set of components,
- in a stable structure,
- which remains unchanged even under faulty conditions,
- and to reflect a particular kind of therapy, namely replacement of components.
From this list, it is obvious that there exist many diagnosis tasks in technical and non-technical domains that are not covered, for instance devices with structural faults or chemical plants in which unexpected reactions occur. Also imagine disturbances in an aquatic ecosystem caused by an unanticipated substance in the sediment that has to be discovered from the observable behavior. Furthermore, once the substance is identified as the cause of the problem, it cannot simply be removed, but rather the situation demands for complex interventions in the system in order to establish external goals again (see the following section for an example of this kind). A detailed analysis reveals that classical component-oriented diagnosis approaches fail to meet the requirements presented by this example (see section 3).

Hence, we need other theories and systems that are more general w. r. t. types of systems, faults, and diagnosis tasks. This thesis presents an answer to this challenge. But rather than developing an additional and different theory, we extend and generalize the existing theories of consistency-based diagnosis, while preserving the approach from "first principles" and the core of the logical foundations (see section 4).

In this thesis, we develop a generalized approach, namely process-oriented consistency-based diagnosis. As a first contribution, a formal reconstruction of a relevant part of a process-oriented modeling paradigm is presented (section 5). This allows us to

- cover systems whose behavior is not established by a fixed set of components in a fixed structure, but rather by processes with complex occurrence conditions (in terms of both structural configurations to be present and quantity conditions to be fulfilled) that interact in a dynamic fashion and may even change the structure by creating new elements - classical component models are reconstructed as a special case,
- identify faults that are not attributable to broken components, but rather caused by unanticipated elements or unexpected interactions, to be discovered dynamically during diagnosis - in a fashion seamlessly integrated with the retraction of default assumptions, allowing for the diagnosis of missing elements, changed variable values, as well as traditional component faults,
- automatically compose a behavior model from a description at the conceptual level (objects, relations, processes) - resulting in a mathematical model that can be used for consistency checking (e. g. of diagnosis hypotheses), which is often a tedious manual task.

In contrast to previous approaches, which usually define a single diagnostic task, we aim at applications (cf. the example above) that require an explicit specification of goals and a differentiation between

- situation assessment: revising an initial model to establish consistency with the observations, and
- therapy recognition: revising an initial model to establish consistency with the goals.

Our second contribution consists in a concise characterization of solutions to these tasks in a non-monotonic logic framework (section 6). A solution is defined as a consistent model with a minimal set of unsupported structural elements (i.e. not created as the effect of an occurring process) and a maximum set of fulfilled default assumptions. Traditional component-oriented diagnosis can be regarded as a specialization, which also sheds light on why and in what respect this task is less complex than the general case. Basing the semantics on minimization of the extension of certain predicates provides elegant definitions, but requires sophisticated algorithms for calculating valid diagnoses.

Therefore, as a third contribution, we provide an algorithmic approach to compute solutions in the sense defined above (section 7). This relies on a transformation of the non-monotonic theory into a monotonic composition of a constraint net representing a superset of all occurring processes - by disregarding part of the occurrence conditions. The creation of a conditional constraint structure prevents the effects of all
collected processes that are not actually occurring (i.e. "inactive"). This construction is proven to be equivalent to the original semantics (see appendix A).

The composed model incorporates a model closure assumption - distributed as "local" closed-world assumptions to places where it has direct consequences - and this turns the following consistency check into a check of completeness at the same time. In the case of inconsistencies, a search for structural revisions is conducted, focused by the additional information contained in the local closed-world assumptions to be retracted.

These algorithms form the basis of an implemented prototype, the Generalized Diagnosis Engine (G*DE) with a flexible architecture and a graphical user interface for modeling and controlling the diagnostic reasoning. The defined graphical representations of modeling constructs facilitate the presentation of examples throughout the thesis. That our approach preserves the core of consistency-based diagnosis shows in the fact that the implementation exploits a commercial software module originally designed for component-oriented diagnosis.

On this basis, we are able to evaluate the new theory in its applicability to the introductory example scenario and to various other classes of problems from different domains. Section 8 presents several extended application examples, concerned with water treatment, medical theory validation and electrical circuits, highlighting different aspects and strengths of the approach and providing an introduction to problem-solving using G*DE.

Finally, we give a short survey of related work, before discussing the advantages and limitations of the proposed theory and implementation, as well as perspectives for future extensions.
2 A Motivating Example

In the following, we present a simplified scenario from a collaborative project in the domain of hydro-ecology and water treatment carried out with partners in Porto Alegre, Brazil. The municipal department of water and sewage in Porto Alegre, DMAE, faces challenges in securing the city's water supply from lakes and reservoirs threatened by an increasing load of organic pollution. The destabilized ecological equilibrium of the small Lomba do Sabão leads to unexpected effects for drinking water generated from raw water captured there: In hot summer days, a distinctly unpleasant metallic taste was observed in the processed drinking water. Analysis of water samples confirmed a high concentration of dissolved iron - above legal and tolerable levels. However, there was no known source of iron - neither in the treatment plant (corroded pipes could be ruled out) nor in the ecosystem itself. The situation is shown in a simple diagram in figure 2.1.

![Diagram of reservoir, pump, tank, and treatment plant showing upper, lower layers, and sediment with observation of metallic taste in drinking water.]

Discussions between environmental experts resulted in a surprising hypothesis emerging as the most likely explanation: There is a high amount of solid iron in the sediment of the reservoir, which was unknown so far. The pH of the bottom layers (hypolimnion) of the Lomba do Sabão typically lies in a medium range and this has almost completely prevented the redissolving of iron into the water body. Suddenly, however, the pH has been significantly lowered, the most likely cause being a local algal bloom triggered by excessive accumulations of nutrients. The unexpected acidic conditions that affected the lower water layers have activated a chemical redissolving process, thus enriching the water with high concentrations of dissolved iron. The iron, ascending to the surface layers, was captured with the raw water intake, and the treatment plant was unable to handle the unexpected high concentrations of iron received and an excessive amount remained in the drinking water, being perceived as an undesirable taste.
After confirming this situation assessment (depicted in figure 2.2), experts discussed possible countermeasures in order to ensure a clean water supply. Certainly, removal of the newly discovered sedimental iron was not an option. Viable alternatives included adding an oxidation agent in the early stages of the treatment process, so that dissolved iron could be removed from the captured water. Also, the application of calcium carbonate as a means of raising the pH of the water body by artificial alkalization was considered. For future occurrences of local algal blooms, algaecides were discussed as a preventive measure, but mainly existing long term plans for the prevention of eutrophication (the accumulation of nutrients in the reservoir) gained additional support. In the end, the application of an oxidation agent seemed to provide both minimum side-effects and the advantage of immediate relieve. All alternatives are shown in figure 2.3.

The example presented above clearly describes a diagnostic task, both in finding potential causes for undesirable system behavior and in discovering ways to influence the system in a positive way. We will refer to these subtasks as situation assessment and therapy recognition and derive solutions for them from a principled approach of process-oriented consistency-based diagnosis. However, it remains to be shown that the "classical" form of consistency-based diagnosis has certain shortcomings and limitations and, thus, fails to meet the requirements as put forth by a large class of problem domains, an instance of which...
is demonstrated in the example. The following section will give a short introduction to the standard approach mentioned.

Based on the extended theory of consistency-based diagnosis developed in this thesis and with its prototypical implementation in the Generalized Diagnosis Engine, G*DE, we will formalize and solve this scenario in section 8.1.1, where also several other application examples can be found.
3 The Standard Approach to Consistency-based Diagnosis

In this section, we briefly summarize the common core of consistency-based diagnosis theories (section 3.1) and systems (section 3.2) in order to confront it with requirements as put forth by the water treatment example (section 3.3), to reveal the built-in assumptions and limitations, and to formulate the goals of a revised theory and implementation.

3.1 The Principle of Consistency-based Diagnosis

Most of the previous work has addressed the task of consistency-based diagnosis from the following perspective (see [Dressler/Struss 1996] for an overview):

- The entities relevant to diagnosis are components $C_i \in \text{COMPS}$, which can be associated with a set of different behavior modes, $\{\text{mode}(C_i)\}$: the correct one, ok($C_i$) (or "normal" in [Reiter 1987] and [de Kleer et al. 1992]) and at least one fault mode ("abnormal", possibly with unspecified behavior).

- A system to be diagnosed consists of a given set of such components, which interact in a way determined by the fixed structure of the system (its blueprint) and are to be scrutinized for faulty behavior.

- The result of diagnosis is an assignment of actual behavior modes to all components.

- The criterion for a proper diagnosis candidate is that the respective mode assignment is consistent with a set of observations, OBS, of the actual system behavior.

If we summarize the theory about the domain (which comprises the description of relevant behaviors of the components of the considered class of devices) and the description of the structure of the particular system and its parameters as the system description SD as in [Reiter 1987], then a diagnosis is defined as a mode assignment that is consistent with the given observations of the system behavior:

$$\text{SD} \cup \{\text{mode}(C_i) \mid C_i \in \text{COMPS}\} \cup \text{OBS} \models \perp.$$  

In general, there are usually many mode assignments satisfying this condition, but they are not equally plausible, and many of them are not of practical interest. If we assume that components fail independently, then if a fault in only one component, $C_0$, is a diagnosis, mode assignments that blame additional components usually do not appear to be of relevance. This leads to a fundamental

Minimality criterion: Consider only minimal diagnoses, i.e. mode assignments for which the set of faulty components is minimal w. r. t. set inclusion.

In the theory of [Reiter 1987], which considers only two modes (abnormal for all "faulty" modes and its negation, ¬abnormal), this corresponds to a preference of (logical) models of the theory that minimize the abnormal predicate. This principle, which is incorporated into the "General Diagnosis Engine", GDE, can be expressed in default logic.
As we will employ default logic for characterizing solutions, as well (section 6.2.2), we will describe a few fundamental features of this non-monotonic logic framework. Default logic ([Reiter 1980]) extends classical first-order logic by adding inference rules of the form

\[ A : B_1, \ldots, B_n / C \]

where \( A, B_i \), and \( C \) are classical formulas. The intuitive reading is: If \( A \) is provable and all \( B_i \) (\( 1 \leq i \leq n \)) are consistent, then \( C \) can be derived. Default schemata are defaults with free variables, and we will follow [Brewka et al. 1997, p. 41] in treating them like the set of all their closed instances, which is slightly different from the more complicated semantics that Reiter attributes to "open defaults".

For a very compact formal definition of the semantics and a discussion of various useful subclasses of default logic, see also [Brewka et al. 1997]. Here, we only mention one of the most important features of default logic, namely that the application of default rules can result in multiple maximal consistent theories called "extensions". Informally, an extension is a deductively closed theory that applies a maximum consistent set of defaults.

While in general, computing extensions of default theories can be very tricky, a useful subset of defaults, the so-called normal defaults of the form

\[ A : B / B \]

can be shown to always produce consistent extensions ([Reiter 1980]). Moreover, useful meta-theoretic properties like cumulativity, semi-monotony and others can be proven for defaults that are also prerequisite-free, i.e. of the form

\[ \text{true} : B / B, \quad \text{or simply} \quad : B / B \]

(see [Brewka et al. 1997, p. 47ff]). We will not go into the details, but we will mostly restrict ourselves to this subclass of defaults.

By using this formalism, the principle that components usually do not break and, thus, minimality of diagnoses can be expressed by a default

\[ : \neg \text{abnormal}(C_i) / \neg \text{abnormal}(C_i) \]

for each component, \( C_i \), with the intuitive meaning "assume the component is not abnormal, unless inconsistent". Minimal diagnoses are then represented by the extensions of the respective default theory. There can be alternative or additional preference criteria:

- **Sherlock** ([de Kleer/Williams 1989]) introduces *mode probabilities* and considers only a set of most probable mode assignments.

- **DDE** ([Dressler/Struss 1994]) generalizes the minimality criterion of [Reiter 1987] through a *preference ordering on the set of modes* for each component. The restriction that a certain mode, \( \text{mode}_i(C_i) \), is considered only if all modes that are strictly preferred over it have been refuted (the set \( \text{pre}(\text{mode}_i(C_i)) \)), is formalized as a set of defaults

\[ (\forall \text{mode}' (\text{mode}' \in \text{pre}(\text{mode}_i(C_i)) \rightarrow \neg \text{mode}'(C_i))) : \text{mode}_i(C_i) / \text{mode}_i(C_i) \]

Preferred diagnoses are obtained from the extensions of the respective default theory.

SD should contain models of the correct behavior of components, but it is not required to provide behavior descriptions for the fault modes. If it does, it may include the assumption that the set of fault modes described is complete, expressed in the finite disjunction
ok(C_i) \lor \text{fault}_1(C_i) \lor \ldots \lor \text{fault}_n(C_i).

and exploit this by exonerating components whose fault modes have all been refuted by the \textit{physical negation rule} ([Struss/Dressler 1989]):

\[ \neg \text{fault}_1(C_i) \land \ldots \land \neg \text{fault}_n(C_i) \rightarrow \text{ok}(C_i) \]

as incorporated in \textit{GDE}' (also [Struss/Dressler 1989]).

\textbf{3.2 Computing Diagnostic Candidates}

An important basis for computing diagnoses in the sense defined above is the representation of SD. It has at least two parts:

- \textit{Model library}: A set of behavior models for component types in the domain, covering at least the correct behavior mode.
- \textit{Structure description}: A set of components and their connections, establishing the particular system under consideration. Mostly, this is organized as "terminals" being shared by connected components.

Additional elements of SD could be preferences, the physical negation rule etc.

A consistency-based diagnosis system has the task to search the space of mode assignments in order to generate all minimal, or, more generally, preferred diagnoses. This is usually done in a hypothesize-and-test manner, which exploits the respective preference criterion and which can be characterized as \"incremental model revision": The search starts with the assignment of the \textit{ok} mode to all components. If a hypothesis (i.e. a particular mode assignment) is refuted, a new set of hypotheses will be generated, which contains the mode assignments that are preferred next to the refuted one, but only after all mode assignments preferred to them have already been tested and refuted. "Testing" diagnostic hypotheses means checking the behavior model corresponding to the mode assignment for consistency with the observations.

The variations mentioned above give rise to different types of systems:

- \textit{GDE} does not use fault models. Hence, except for checking consistency of the model of correct behavior, there is no test step, and no fault mode can be refuted.
- \textit{Sherlock} involves fault models and tests mode assignments in the order of their a posteriori probabilities calculated from the observations. It contains an unknown fault mode for each component (with no associated behavior model), which has a low probability and, hence, will be considered only late or never in the search process.
- \textit{GDE}' employs fault models and the physical negation rule to exonerate components.
- \textit{DDE} traverses the mode preference lattice guided by the preference default rules.

If a mode assignment is refuted, its successors can often be generated in a focused manner, because usually it is possible to localize the source of an inconsistency to some degree. If the starter of a car does not work, we do not suspect the windshield wipers, because they do not interfere with this function. This means that the starting point for hypothesizing new mode assignments will be given by sets of models of component modes that together are inconsistent with the observations but form only a subset of the entire model, so-called \textit{conflicts}. Such a conflict states that at least one of the involved components is not in accordance with its model of the respective behavior mode.
Conflicts can be computed by dependency-based prediction, where the consequences of observations and known facts are generated and annotated with all modes that are used in predicting new values or other facts. Inconsistencies then easily yield conflicts and these can be used to derive minimal diagnoses. For instance, if the test of the model of correct behavior has delivered two conflicts

\[\neg \text{ok}(C_1), \neg \text{ok}(C_2), \neg \text{ok}(C_1), \neg \text{ok}(C_3)\],

then two diagnoses can be obtained that are minimal (w. r. t. set inclusion):

\[\neg \text{ok}(C_1), \neg \text{ok}(C_2), \neg \text{ok}(C_3)\],

and the next mode assignment hypotheses need to consider fault modes of \(C_1\) or \(C_2\) and \(C_3\) combined with the \(\text{ok}\) modes for all other components.

### 3.3 Revisiting the Example

If we revisit the example introduced in the previous section, we notice that diagnosis of devices made up of components in a fixed structure is much simpler than solving problems in water treatment plants or ecological systems. Although components such as pumps, pipes, and tanks are involved in water treatment, it is not their malfunctioning that is causing the problem. Also it is not the case that any of the chemical, mechanical, or biological processes involved do not perform well.

What we are really looking for are additional system constituents (at least in the sense that they were not assumed nor needed in the initial system description), namely sedimental iron and an algal bloom. Still, one can employ an approach of retracting default assumptions when reasoning about the unexpectedly low pH for the bottom of the reservoir.

The difference to standard diagnosis approaches becomes even clearer when looking at remedies for the iron spoiling the drinking water. There are no "components" to replace, nor is there a tractable way to simply remove the iron from the sediment. Rather, therapy recognition has to find another achievable path to a desirable system state, e. g. by introducing another additional element, namely an oxidation agent or calcium carbonate.

Our small diagnostic problem also reveals that simply checking the system model against the observations is a specialized and narrow perspective: if our model includes the reservoir with algal bloom and high iron concentrations and the usual water treatment processes (without an additional oxidation agent), the observation of bad drinking water quality is perfectly consistent with this model, it is just unwanted. While the original theory implicitly assumed that the system model carries the "gold standard" ("if all components work properly, the overall behavior is the desired one"), we now realize: what is crucial is the inconsistency of the behavior with our intention or goals, i. e. something that is external to the model. This is not only obvious for the processes in nature - algal bloom may be undesirable, but it is a natural process, nothing "faulty" - it also applies to technical processes.

We summarize in what respect the "classical" theories and systems of consistency-based diagnosis are too narrow and, as a result, fail to provide a solution to many diagnostic problems in the environmental domain, but also in various technical applications:

- We could not call iron redissolving a fault. Natural processes do not break or fail like components. This means: The relevant constituents of the system do not have fault modes.
- It is not the case that one of the constituents of our original system description can be blamed for the inconsistency with the observations. The reason is an additional, unanticipated constituent, namely
sedimental iron, we were not aware of. This means: a revision of the system description cannot be confined to a set of given constituents (the "components").

- Also, for finding an appropriate treatment, changing the "mode" of the given constituents, e. g. by replacement of a broken component, is not the issue. One has to find actions that, again, might expand the entire system, e. g. by introducing algaecides or an oxidation agent.
- There are no "failures of nature". Algal blooms are not a fault, even though we might want to avoid them. A given phenomenon may be perfectly consistent with the observations, while inconsistencies arise only with our goals and intentions. This means: the classical diagnosis task, based on inconsistencies between the model and the observations, is to be explicitly split into situation assessment (using the observations) and therapy recognition (taking the goals into account).

This analysis determines requirements and goals for an attempt to develop a new theory of diagnosis based on first principles: It has to

- cover systems whose behavior is not established by a fixed set of components connected in a fixed structure, but which comprise processes that interact in a dynamic fashion and may even create new elements, e. g. dissolved iron from solid iron,
- be able to identify the occurrence of unanticipated objects and processes as causes of misbehavior, and
- make explicit the target behavior of a system, thus determining whether or not a certain behavior is considered to be misbehavior, which enables it to
- distinguish between contradictions between a behavior model and the observation on the one hand and between a model and a desired behavior on the other hand. It thus has to incorporate the ultimate goal of diagnosis, namely to re-establish the intended function of a system.

These objectives require revisions in both the modeling formalism and the diagnosis theory and algorithm. In the next section, we will outline the foundation of our solution, which is derived by first isolating the general core of consistency-based diagnosis and then formulating the above requirements in a more systematic and technical way.
4   Foundations of the Generalized Theory of Consistency-based Diagnosis

If we strip off the specific orientation towards component-oriented diagnosis, what remains of traditional theories of consistency-based diagnosis is the plausible and powerful idea of

- regarding diagnostic problem solving as a search for models of a system that are consistent with some criterion which is external to the model, in particular the observations, and
- to perform this search by incrementally revising some initial model by local and, in some sense, minimal changes.

This requires two basic elements in the theory and in any system implementing it:

- *Model-based prediction and consistency checking*: We need a formal, usually mathematical, model as well as an algorithm that uses this model to derive more information about the behavior of the system and to check this against the respective criterion. If this criterion is a set of observations about the system behavior, then models that pass the consistency check are candidates for assessing the observed situation properly. If we introduce a specification of the desired system behavior, GOALS, then the consistent models represent possible remedial actions applied to the system (or its "re-design"). The two versions are depicted in figure 4.1.

- *A model revision step* which generates revised model hypotheses from those found inconsistent with the respective criterion using a library or *domain theory*, i.e. a repository of potential model fragments, which has to include possible disturbances (for situation assessment) and interventions (for therapy recognition).

This establishes the overall process of a consistency-based problem solver as depicted in figure 4.2.

![Figure 4.1: Prediction and consistency check for situation assessment (top) and therapy recognition (bottom)]
A closer analysis will allow us to state the objectives from the previous section as requirements on the elements of this problem solving approach.

First, we have to reflect the nature of model hypotheses and their generation by revision: In human reasoning and in model-based systems, this involves concepts of the physical systems and principles in the domain, such as types of objects and their properties, the structure of systems, faults, diseases, and human interventions. In other words, the resulting model hypothesis is stated at a conceptual level, while the model required for prediction and consistency checking has to be a mathematical behavior model stated in terms of (differential) equations, constraints or some other formalism - model revision and prediction happen at different modeling levels.

Bridging this gap is necessary, and if the entire model search is to be automated, it cannot be done (as usual) by humans, but has to be performed automatically. This sets the requirement for another element: A model composer automatically generates an executable mathematical model from a conceptual specification of a physical system using the domain theory (see figure 4.3).

This establishes a strong requirement on the modeling formalism:

- **Conceptual modeling**: The model fragments in the domain theory need an explicit representation of the conceptual modeling level.

- **Compositional modeling**: The fragments of the mathematical behavior model need to be associated with a conceptual description of the conditions under which they are valid and have to be included in the behavior model. This forms the basis for an algorithmic solution to the composition of the overall behavior model used for prediction and consistency checking.

In component-oriented modeling, these requirements are satisfied in a straightforward way: The core concept is the behavior mode of a component, and a behavior model of a system with an enumerated set of components and their interconnections is obtained as a conjunction of the respective behavior model fragments with those variables identified that are shared by different individual components (say, voltage...
and current for connected electrical components). Basically, it means to include the constraints describing the behavior under the various modes:

\[
\text{mode}_1(C_i) \rightarrow \text{constraint}_{i1} \\
\text{mode}_2(C_i) \rightarrow \text{constraint}_{i2} \\
\vdots
\]

As the water treatment example illustrates, our generalized theory has to provide more than this: Model composition has to be able to complete a given partial description of system in terms of known objects, their relations and properties by *deducing* the existence of certain processes (such as redissolving of iron). It also has to be able to identify the existence of a substance as the effect or precondition of processes (e.g. solid iron in the sediment as a condition for redissolving into the water body). Qualitative Process Theory ([Forbus 1984]) provides this capability and the inspiration for our solution.

While the model composer establishes the step from a conceptual to a mathematical model, the *conflict generator* has to turn inconsistencies detected at the mathematical level into statements at the conceptual level as a starting point for model revision, thus completing the overall picture of consistency-based problem solving (figure 4.4).

![Figure 4.4: Problem solving based on model revision](image)

For component-oriented diagnosis, conflicts comprise component modes only, and the space of possible revisions is spanned by the finite set of modes of each component type,

\[
\text{Component}(C_i) \rightarrow \text{mode}_1(C_i) \lor \ldots \lor \text{mode}_n(C_i)
\]

for an enumerated set of components:

\[
\text{Component}(C) \rightarrow (C = C_1) \lor \ldots \lor (C = C_m)
\]

and model revision is guided by the preference criteria discussed in section 3.

The objectives of the process-oriented diagnosis approach impose additional requirements on the revision step. This step and, hence, the conflicts should not only involve variable values, objects, and relations mentioned in the initial model, it also has to be able to hypothesize additional objects and relations. Most predictions rely on a certain closure of the model establishing a usable form of the behavior model. By making this closure explicit, it can be refuted and revised. Even more important, this revision has to be performed in a focused manner, which is obviously much more complex than in the component-oriented case, where only simple mode switches for suspect components are required. Also, the minimality
principle stated above cannot be directly employed for focusing the search for solutions in the more
general sense.

Finally, conflict generation imposes new requirements on the algorithmic solution. In the composition
step, model fragments are associated with configurations of conceptual units, and this association has to
be recorded in order to obtain meaningful conflicts later. Model composition is no longer a one-to-one
association of behavior model fragments to component modes, but involves potentially complex
inferences. Therefore, determining the ultimate (revisable) units that together justify the existence of a
particular model fragment becomes non-trivial, as well.

Consequently, to be able to provide a firm ground for our logical theories of diagnosis and, in particular,
for conflict generation, we first need to formalize the chosen approach to modeling and model
composition. This is why we will continue with a logical reconstruction of process-oriented modeling -
which will be done in a way that can also be viewed as a generalization of component-oriented modeling
approaches.
5 The Modeling Approach

As mentioned before, in Reiter's classical paper "A Theory of Diagnosis from First Principles" ([Reiter 1987]), the system description (SD) is defined as simply "a set of first-order sentences". While this is a conveniently general definition, open to largely different modeling approaches, it is not restrictive enough to exclude theories unsuited for GDE-style diagnosis (which relies on the independence of the introduced assumptions, for instance) or to prevent the usage of modeling schemes subverting all benefits of the principled reasoning. Since then, modeling methodologies, theories of model structure and revision, and of model abstraction and transformation, have been subject to intensive research in model-based diagnosis ([Dague et al. 1987], [Hamscher 1991], [Falkenhainer/Forbus 1991], [Struss 1992]). We argue that it is necessary to also provide a proposal for the overall structure of models to be used as well as formal definitions for modeling primitives.

In this section, we will shortly describe the key features of Qualitative Process Theory, which forms a basis for flexible and expressive behavior models (section 5.1). We also discuss, how the component-oriented modeling paradigm can be seen as a specialization of the more general process-oriented one (section 5.2). We continue with a presentation of our formalization of a relevant part of QPT (section 5.3).

5.1 Process-oriented Modeling

While the component-oriented modeling paradigm supports the features of compositionality, re-usability and independence of context, reasoning about domains with a dynamically changing structure (like hydro-ecology) requires more powerful and flexible modeling capabilities. Dynamic changes in the constituents of a portion of an ecosystem can trigger largely different phenomena contributing to the overall system behavior, even influencing the configuration itself that has been its very cause. It is largely impossible to impose a rigid structure on the behavioral model fragments of an hydro-ecological system for instance, as one can do in representing a circuit layout. Also structural elements like substances can be created by chemical reactions or be destroyed again.

To account for systems of such fundamentally different dynamics, process-oriented models are a promising way to capture the laws of physics (and biology, chemistry and other disciplines) - on a conceptual level. Forbus has developed a fundamental theory for using qualitatively abstracted process models for highly dynamic domains, the "Qualitative Process Theory", or QPT [Forbus 1984]. One of the principles of QPT is called "structure-to-behavior reasoning", indicating that modeling is based upon an explicit representation of structure in terms of objects (and predicates and relations). As a second layer, phenomena that can occur are modeled as processes or "views". The occurrence of processes is determined by the presence of certain object configurations and their properties.

Objects are typed and quantities are associated with individual objects of each type to represent properties that can be changed by the phenomena modeled. Examples are the temperature and the amount of a liquid. Predicates are used for properties that are exogenous to the modeled system, e. g. the capability of a container to hold liquid or conditions that depend on a particular control action. The representation of quantity values is an important part of QPT, which will be mentioned here only briefly, since we take a different approach. Forbus is serious about a widely adopted principle of
Qualitative Reasoning, namely "making only relevant distinctions". This leads him to the construction of a partially ordered space of symbolic values for each quantity, the so-called *quantity space*. Starting from a set of initial distinctions, further values can be incorporated into a particular quantity space by reasoning about their relevance for the context of the quantity. An elaborate theory of inequality reasoning has been developed, including mechanisms for the discovery of "corresponding values" across different quantity spaces.

For behavioral descriptions, QPT provides the constructs *process* and *individual view*. The theory is deterministic in stating that if and only if there is a configuration of objects satisfying a specified set of conditions, the process or view will take effect. Furthermore, the "sole mechanism assumption" is made, asserting that every change in a quantity value is due to a process being active.

Conditions for processes and views can be expressed in terms of object types ("individuals"), predicates and inequalities between quantity values ("preconditions" and "quantity conditions" in QPT, the latter can be determined from the variable values predicted). Interestingly, Forbus separates instantiation from activation of processes and views from the very beginning, as an integral part of the definition. The theory requires a process or view instance to be present for each set of objects matching the specification of "individuals", but its activity depends on the preconditions and quantity conditions. We will discuss activity variables as a particular design decision, when regarding reasoning about process models in section 7.1. It is, however, worth mentioning that defining the semantics of the modeling primitives can be done more concisely without allowing processes that are present, but inactive.

The effects of processes are given as relations and influences between the quantities of the objects involved. Relations include qualitative proportionalities, i.e. monotonic dependencies that are not further specified, except for possible corresponding values. Influences relate two quantities in a causal relationship. A process description defines direct influences between quantities, which are usually provided with semantics that hold for the complete (closed) model. Forbus states that the derivative of a quantity equals the sum of the direct influences on the quantity. This is also termed the "linear combination assumption". An indirect influence is defined to be present when a quantity is in a functional relationship with a directly influenced quantity.

*Individual views* are designed to represent the existence of contingent objects, a typical example being a "contained liquid", which requires a liquid, a container and a spatial containment relationship to exist. A view establishes relations (but not influences) as effects, which can include qualitative proportionalities between quantities, but can also define the existence of objects. It is not entirely clear, whether these objects can in turn participate as "individuals" in other process or view instances, or only be named explicitly in a set of preconditions (as a kind of complex predicate), since Forbus does not discuss the intricate issues of the circular dependencies of process instances requiring object existence, which in turn depends on process activity. The examples given in the original paper [Forbus 1984] are suggesting different mechanisms of object creation (pages 99 and 107), while formal semantics are missing.

The final constituent of a QPT model is a *scenario description* that names a set of objects and the extensions of certain predicates, along with inequalities about quantity values. Given such a description, a reasoning system such as QPE, the Qualitative Process Engine ([Forbus 1990]), can carry out different kinds of inferences with prediction, simulation, and explanation generation being the most common ones. So the theory is not tailored for a particular task, but rather provides general means of deriving knowledge about the behavior of complex systems.
5.2 Generalizing Component-oriented Modeling

As discussed in the previous section, the process-oriented model paradigm is suitable for modeling highly dynamic systems that are not accessible to component-oriented approaches for the lack of a fixed structure and dynamically changing patterns of activity. Nevertheless, for most technical devices it is more appropriate to divide them into functional units with a given set of connection points, rather than seeking to represent the detailed processes of electrical conductance or hydraulic pressure propagation across components. So, are we left with two distinct methods for two categories of systems and have to resort to "hybrid" modeling for systems comprising both hydraulic controllers and hydro-chemical reaction tanks? Fortunately not.

When taking a sufficiently abstract perspective, we can discover many features of the process-oriented approach in a very specialized way in component-oriented reasoning. The separation of component (type) models and the connection structure can also be seen as an instance of "structure-to-behavior" reasoning. With a carefully designed process-oriented model, one can make sure that the right model fragment instances are collected and activated - in the simplest case, the respective preconditions are always true. The quantity effects are restricted to constraints, since the complex influence resolution mechanisms are not needed, for there will be no interactions created or removed dynamically. See section 5.3.3.1, as well as the examples in section 8.3 for a reconstruction of component models within our framework. In this sense, components can be treated as a special case of the carefully designed modeling approach presented here, which leads us to name the basic behavior model fragments neutrally behavior constituents rather than "processes" or "component type models".

In particular, this generalization gives us the opportunity to use components interacting with processes. An example comprising electrical circuits and thermodynamic interactions can be seen in section 8.3.1.

5.3 Formally: The Modeling Approach

In this section, we formally describe the modeling approach. After an overview of the general language design, all modeling primitives and their semantics are presented in first-order logic. Additionally, we introduce a graphical specification language, as also employed in the user interface of our prototypical implementation (cf. section 7.3).

The basic building blocks of the modeling language are the Domain Theory and the Situation Description. The **Domain Theory** consists of an **Ontology**, **Quantities**, and a set of **Behavior Constituent Types**, as well as the **Basic Axioms**. The **Ontology** defines the entities that can be used in representing structure: object types and structural relations. **Quantities** of defined types can be associated with object instances. **Behavior Constituent Types** are an abstraction of process and behavior models, as mentioned in section 5.2. They are the fundamental elements of behavioral descriptions in deterministic laws of the form:

\[
\text{IF Structural Conditions AND Quantity Conditions} \\
\text{THEN Structural Effects AND Quantity Effects.}
\]

The **Basic Axioms** are the formal representation of the semantics of the modeling primitives, e. g. the laws governing type hierarchies and quantity values assignments, or the rules for behavior constituent effects. These axioms have to be formalized for proof of correctness or support of program design verification (see appendix A), but we will usually explain all laws directly with the modeling primitives they refer to. The motivation for declaring a separate "section" of the model for the Basic Axioms is to distinguish
clearly between the statements made by a "user" of the diagnostic system to describe (a class of) real-world systems at hand and the statements that hold universally for all systems described using this modeling paradigm. Typically, the former are existentially quantified sentences in the first-order theory, while the latter are universally quantified (see figure 5.3 below).

Finally, a **Situation Description** is similar to what QPT calls a scenario. It consists of object instances (and relationships between them) and initial values of associated quantities, all possibly supported by retractable user-defined assumptions.

In figure 5.1, an overview of the modeling sections is shown:

<table>
<thead>
<tr>
<th>Domain Theory</th>
<th>Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Object Types (hierarchy)</td>
</tr>
<tr>
<td></td>
<td>Relations (properties)</td>
</tr>
<tr>
<td></td>
<td>Quantities (types, associations)</td>
</tr>
<tr>
<td>Behavior Constituent Types</td>
<td>Structural Conditions (SC: objects and relations present)</td>
</tr>
<tr>
<td></td>
<td>Quantity Conditions (QC: constraints on quantities)</td>
</tr>
<tr>
<td></td>
<td>Structural Effects (SE: objects and relations created)</td>
</tr>
<tr>
<td></td>
<td>Quantity Effects (QE: constraints and influences)</td>
</tr>
<tr>
<td>Basic Axioms</td>
<td></td>
</tr>
<tr>
<td>Situation Description</td>
<td>Objects (instances)</td>
</tr>
<tr>
<td></td>
<td>Relation Tuples (between object instances)</td>
</tr>
<tr>
<td></td>
<td>Quantity Value Assignments</td>
</tr>
</tbody>
</table>

**Figure 5.1: Overview of the Modeling Sections**

The distinction between the system description (SD) and the observations (OBS) made in classical diagnostic theories is more sophisticated in our case. In figure 5.2, the specificity of the sections is depicted. The domain theory is specific to a class of systems in a common "domain", while the situation description has a system specific part, assumed to be true for the entire history of the system or at least the relevant part of it, and a situation specific part. For instance, quantity value assignments are used both for invariable parameter specifications (e.g. the volume of the water body under consideration) and for observation of specific system states (e.g. the temperature at a certain time point). But also the set of objects is not necessarily fixed, as substances might be created dynamically. The basic axioms are completely generic and, thus, invariably the same for any models expressed in the described modeling language.

This is to say that not all parts of the model can be specified by the "user" of an implemented system for reasoning about such models. Most notably, the basic axioms can be "hard-wired" into the reasoning system in the form of algorithms (e.g. for object type classification and matching). We propose to provide a more or less comprehensive basic ontology for the user to refine, if she aims at engineering the modeling foundations, at all. A set of generic behavior constituents, or "templates" might be extended by the experienced user, while the situation description will be easier to handle for a user who wants support for situation assessment, diagnosis, or therapy. We will mention some issues of interface design in section 7.3. This will result in a picture as in figure 5.3.

**Figure 5.2: Specificity of the Modeling Sections**
Figure 5.4 shows the parts of the model that are revisable by reasoning. Object instances can be introduced - or assumed ones can be retracted. Likewise, some relation instances ("tuples") will be open to revision. The values of quantities are the most common case of information to be derived (predicted), rather than observed. As a remark, the efficiency of the consistency-based diagnosis approach depends crucially on the revisables to constitute a comparatively small part of the overall model. Generic model synthesis and domain theory revision is beyond the scope of this thesis.

In the following, we take a look at each of the modeling sections in turn and describe the primitives that can be used, along with logical definitions and graphical representations. Section 5.3.1 handles the Ontology, section 5.3.2 the Quantities. In section 5.3.3 the central concept of behavior constituents is formalized. The significance of basic axioms is summarized in section 5.3.4 and section 5.3.5 discusses the important closure of the domain theory. Finally, the situation description is presented in section 5.3.6.

5.3.1 Ontology

Structure descriptions are composed of object instances and relations between them. All behavior can be observed through the change of values of quantities associated with these objects. The Ontology section of the model defines the classes of structural elements allowed.

5.3.1.1 Object Types

Firstly, a set of object types and a hierarchical taxonomy of these object types are defined. Each type can have more than one supertype, but, of course, there should be no circularity, i.e. an object type should not be the supertype of any of its own supertypes.

Syntax: The user can provide a set of the following predicates - with arbitrary constants, \(ot\) and \(ot'\), for object types:

\[
\begin{align*}
\text{ObjectType}(ot) \\
\text{IsA}(ot, ot')
\end{align*}
\]
Semantics: The IsA predicate is defined for object types only and is transitive:

\[ \text{IsA}(ot_1, ot_2) \rightarrow \text{ObjectType}(ot_1) \land \text{ObjectType}(ot_2) \]
\[ \text{IsA}(ot_1, ot_2) \land \text{IsA}(ot_2, ot_3) \rightarrow \text{IsA}(ot_1, ot_3) \]

The important part is that objects (defined in the Situation Description) that are assigned an object type using the predicate IsOfType, also belong to all supertypes of the given object type, i.e.

\[ \text{IsOfType}(o, ot) \land \text{IsA}(ot, ot') \rightarrow \text{IsOfType}(o, ot') \]

This, and the inheritance of associated quantities (see below) are the rationale for introducing object type hierarchies.

Remarks: In these and all following axioms, all open variables are implicitly quantified universally.

With these formalizations, we aim at giving evidence that the semantics of the modeling language are well-defined - and that the designed algorithms compute exactly the characterized solutions. However, the set of axioms will not be complete, as for instance we omit the fundamental laws for non-overlapping extensions of "types" in the style of

\[ \forall x \text{ObjectType}(x) \rightarrow \neg \text{Relation}(x) \land \neg \text{Object}(x) \land \neg \text{Quantity}(x) \land \ldots \]
\[ \forall x \text{Relation}(x) \rightarrow \neg \text{Object}(x) \land \neg \text{Quantity}(x) \land \ldots \]

We use logic in describing the semantics, but advocate graphical user interfaces for implemented reasoning systems to specify, specialize and modify models. Our prototypical implementation (see section 7.3) integrates a commercial diagram editor that can be used for most of the user input, including the complex specification of the domain theory. All of the examples in this section, as well as in section 8 are direct exports from this editor. This relieves the user from thinking about predicates, and rather enables her to focus on building a diagram of the object type hierarchy, for instance.

Example: See figure 5.5, where several object types are defined in a multiple inheritance hierarchy:

![Diagram of object type hierarchy](image)

Figure 5.5: Example of object type hierarchy
Substance is declared a SpatialEntity with subtypes SuspendedSubstance, DissolvedSubstance, and SolidSubstance and their respective subtypes. Note that DissolvedIron is a subtype of both DissolvedSubstance (so it can occur in relations specific for dissolved substances) and Iron (so in other respects it can be treated similar to SolidIron).

This is a part of the formalization of the introductory example (see section 2) as carried out in section 8.1.1. Also, all other examples in section 8 make use of rather complex object type hierarchies, often involving multiple inheritance.

5.3.1.2 Relations

Secondly, the set of relations that can be used to relate objects to each other is defined. For each relation, that has to be of finite arity, the object types that can participate are declared.

Syntax: This can be written as

\[
\text{Relation}_n(\text{rel}) \\
\text{RelationParameterTypes}_n(\text{rel}, \text{ot}_1, \ldots, \text{ot}_n)
\]

with \(n\) denoting the arity of the relation.

Additionally, we allow the special properties of "symmetry", "completeness", "uniqueness" and "functionality" to be specified for binary relations, \(\text{rel}\).

\[
\text{RelationSymmetric}(\text{rel}) \\
\text{RelationComplete}(\text{rel}) \\
\text{RelationUnique}(\text{rel}) \\
\text{RelationFunction}(\text{rel})
\]

Semantics: First, let's look at the parameter types for the various predicates:

\[
\text{RelationParameterTypes}_n(\text{rel}, \text{ot}_1, \ldots, \text{ot}_n) \rightarrow \text{Relation}_n(\text{rel}) \land \text{ObjectType}(\text{ot}_1) \land \ldots \land \text{ObjectType}(\text{ot}_n)
\]

\[
\text{RelationSymmetric}(\text{rel}) \rightarrow \text{Relation}_2(\text{rel})
\]

\[
\text{RelationComplete}(\text{rel}) \rightarrow \text{Relation}_2(\text{rel})
\]

\[
\text{RelationUnique}(\text{rel}) \rightarrow \text{Relation}_2(\text{rel})
\]

\[
\text{RelationFunction}(\text{rel}) \rightarrow \text{Relation}_2(\text{rel})
\]

We employ a semi-formal notation with ellipses ("...") for a set of axioms with a varying (but finite) number of parameters.

Next, we define the uniqueness of the defined parameter types:

\[
\text{RelationParameterTypes}_n(\text{rel}, \text{ot}_1, \ldots, \text{ot}_n) \land \text{RelationParameterTypes}_n(\text{rel}, \text{ot}_1', \ldots, \text{ot}_n') \rightarrow \text{ot}_1 = \text{ot}_1' \land \ldots \land \text{ot}_n = \text{ot}_n'
\]

Symmetry of a binary relation is defined as

\[
\text{RelationSymmetric}(\text{rel}) \land \text{RelationParameterTypes}_2(\text{rel}, \text{ot}_1, \text{ot}_2) \rightarrow ((\exists rt \text{ RelationTuple}_2(rt, \text{rel}, \text{ot}_1, \text{ot}_2)) \leftrightarrow (\exists rt' \text{ RelationTuple}_2(rt', \text{rel}, \text{ot}_2, \text{ot}_1)))
\]

This makes use of a predicate RelationTuple\(_n(rt, rel, o_1, ..., o_n)\) expressing that there is a relation tuple, \(rt\), of the given relation, \(rel\), connecting objects, \(o_1\) through \(o_n\), in the given order. It obeys the following restriction:

\[
\text{RelationTuple}_n(rt, rel, o_1, ..., o_n) \land \text{RelationTuple}_m(rt', rel', o_1', ..., o_m') \rightarrow (rt = rt' \leftrightarrow n = m \land rel = rel' \land o_1 = o_1' \land \ldots \land o_n = o_n')
\]
With "completeness" of a binary relation, rel, defined for the object types ot1 and ot2, we denote the property that for any instance of ot1, there has to be an instance of ot2, related via rel. Formally,

\[\text{RelationComplete}(rel) \land \text{RelationParameterTypes2}(rel, ot1, ot2) \rightarrow \forall o1 \text{ IsOfType}(o1, ot1) \exists o2 \text{ IsOfType}(o2, ot2) \exists rt \text{ RelationTuple2}(rt, rel, o1, o2)\]

"Uniqueness" of a relation indicates that the relation is a partial mapping, if reversed. Namely, for each object instance in the second parameter of the relation form there is at most one instance that can occupy the first parameter position:

\[\text{RelationUnique}(rel) \land \text{RelationParameterTypes2}(rel, ot1, ot2) \rightarrow (\text{RelationTuple2}(rt1, rel, o1, o2) \land \text{RelationTuple2}(rt2, rel, o1', o2) \rightarrow o1 = o1')\]

And, finally, we have the complement of uniqueness:

\[\text{RelationFunction}(rel) \land \text{RelationParameterTypes2}(rel, ot1, ot2) \rightarrow (\text{RelationTuple2}(rt1, rel, o1, o2) \land \text{RelationTuple2}(rt2, rel, o1, o2') \rightarrow o2 = o2')\]

Remarks: A typical usage of the uniqueness property is the modeling practice to use a single object instance for a uniformly mixed substance in a compartment or tank, e. g. dissolved iron in a water compartment. The uniqueness property ensures the singularity of this object w. r. t. the location, so that its associated "concentration" quantity (see figure 5.7 below) can be used as a representation of the (unique) iron concentration in the specified volume of water. This is the preferred way to model spatially distributed parameters using our ontology.

The general idea behind relation properties is the disambiguation of situation descriptions given or derived by reasoning. Each unnecessary ambiguity complicates the reasoning process significantly, since we might have to create "multiple worlds", reason in each of these worlds and retain only the conclusions that can be proven to be consistent in all worlds. Our approach is essentially to avoid this kind of structural ambiguity wherever possible.

Example: Again, we illustrate an example of relation definitions using a graphical notation (see figure 5.6).
Four binary relations, located-at, dissolved-in, connected-to and connected-from, are defined by enumerating the object types of their parameters, e. g. SpatialEntity and SpatialLocator for located-at. Additionally, the relations are supplied with some properties. For the latter two relations this means that a Pump is to be connected to exactly one source LiquidLocator and one destination LiquidLocator: completeness ensures that there is at least one, functionality that there is not more than one.

5.3.2 Quantities

A second part of the Domain Theory is the definition of quantity types and the association of quantity instances to object instances. More precisely, for an object type one can define a set of "quantity roles" together with their quantity types. For each object, the respective set of quantities of the given type (referred to via its object role) can be used in specifying observations, conditions, or effects of behavior constituents.

Syntax: A quantity type is characterized by

\[ \text{QuantityType}(qt) \]

For an object type, \( ot \), the user can specify associations via

\[ \text{QuantityRole}(qr) \]
\[ \text{QuantityAssociationObjectType}(ot, qr, qt) \]

Remarks: We do not formalize quantity type domains, which could be defined by enumerating a finite set of possible values or by using pre-defined domain symbols. We will see in a later section (7.2) that the generalized diagnostic reasoning proposed here is open to arbitrary symbolic or even numeric calculations, if a constraint system capable of handling the respective constraints is available.

Note that Forbus uses the term "quantity type" in a different way in QPT. For all practical purposes, it corresponds to what we call a "quantity role".

A restriction in our modeling approach is that quantities cannot be associated with relations or groups of objects, but rather with single object instances only. Our main argument is that a concept worthy of carrying a quantity can also be promoted to be a full object. Using this procedure of "reification" one can almost always produce an equivalent model with object quantities only.

Semantics: There is exactly one quantity for each object of a given type and for each quantity role specified for this type:

\[ \text{IsOfType}(o, ot) \land \text{QuantityAssociationObjectType}(ot, qr, qt) \rightarrow \]
\[ \exists qu \text{ Quantity}(qu) \land \text{HasQuantityRole}(qu, o, qr) \]
\[ \text{HasQuantityRole}(qu_1, o_1, qr_1) \land \text{HasQuantityRole}(qu_2, o_2, qr_2) \rightarrow (qu_1 = qu_2 \leftrightarrow o_1 = o_2 \land qr_1 = qr_2) \]

Remember that IsOfType relies on the transitivity of Isa (see above). Thus, associations of quantities will be inherited from supertypes - but the individual quantities are separate: a quantity takes on exactly one role for a single object (the second axiom of the above) - this uniqueness of quantities will be important for defining influence resolution (see section 5.3.3.6).
Furthermore, quantities are assigned exactly one value. We don't formalize quantity type domains here, so we omit the condition that the value has to be a member of it.

\[ \forall qu \text{ Quantity}(qu) \rightarrow \exists v \text{ Value}(v) \land \text{HasValue}(qu, v) \]
\[ \text{HasValue}(qu, v_1) \land \text{HasValue}(qu, v_2) \rightarrow v_1 = v_2 \]

**Example:** See figure 5.7 for an example of quantity associations. A quantity of type NonNegative (to be defined elsewhere) is associated with the object type Substance. The respective quantity role is named concentration. Similarly, quantities pH and volume are associated with each LiquidLocator.

![Figure 5.7: Example of quantity associations](image)

### 5.3.3 Behavior Constituent Types

All observable behavior, as expressed in the values of quantities, is determined by the set of occurring behavior constituents - this is a variation of the "sole mechanism assumption" mentioned in section 5.1. Behavior constituents are typed and a behavior constituent type is basically a deterministic law of behavior made up of structural conditions (SC), quantity conditions (QC), structural effects (SE), and quantity effects (QE) with the following semantics:

For each configuration of objects and relations matching the structural and quantity conditions, the structural and quantity effects will result.

A formalization of this rule will be derived in the following. Consider the diagram of a behavior constituent type definition in figure 5.8. The intuitive meaning is that iron can be dissolved from the sediment into the water layer directly above, provided the pH is low enough and there is a significant concentration of solid iron. All modeling primitives used, together with the laws of occurrence and influence resolution will be described in the next sections.

![Figure 5.8: Example of behavior constituent type definition: IronRedissolving](image)
5.3.3.1 Structural Conditions

The structural conditions (SC) of a behavior constituent type are expressed as a set of object instance templates and relation tuple templates between them. An object instance template consists of an object type and an object role. It can match any object of the given type (which includes all of its subtypes) that will then be bound to the given role. These roles can appear in a relation tuple template, identifying one of the relations defined in the Ontology section. This template matches if and only if there is a relation tuple of the given relation between the objects bound to the respective object role parameters. The complete condition matches a configuration if and only if all object and relation templates match consistently.

Syntax: We use the following predicates for expressing that the existence of an object or relation tuple is part of the structural conditions for a behavior constituent type \( \text{bct} \):

\[
\begin{align*}
\text{StructuralConditionExObject}(\text{bct}, \text{or}, \text{ot}) \\
\text{StructuralConditionExRelationTuple}_n(\text{bct}, \text{rel}, \text{or}_1, \ldots, \text{or}_n)
\end{align*}
\]

where \( \text{or} \), and \( \text{or}_1 \) through \( \text{or}_n \) denote object roles, \( \text{ot} \) an object type and \( \text{rel} \) a relation defined in the Ontology section.

Example: Graphically, one can specify structural conditions by including a configuration template consisting of object types and relations. The shaded area in figure 5.9 indicates the structural conditions of the IronRedissolving behavior constituent type:

![Figure 5.9: Structural conditions of IronRedissolving](image)

There is to be a solid layer (role name \( \text{aSediment} \), type \( \text{SolidLayer} \)) directly below a water layer (\( \text{aLayer} \), to the right). Additionally, some iron has to be present in the sediment (role \( \text{IronSediment} \) of type \( \text{SolidIron} \), relation fixed-in).

Semantics: For defining the semantics of the structural conditions, we employ mappings of object roles to objects. Firstly, we characterize such mappings and then require their existence for each possible combination of objects:

\[
\begin{align*}
\text{Maps}(m, \text{or}, o) & \rightarrow \text{Mapping}(m) \land \text{ObjectRole}(\text{or}) \land \text{Object}(o) \\
\text{Maps}(m, \text{or}_1, \text{or}_2) \land \text{Maps}(m, \text{or}_1, \text{or}_2) & \rightarrow o_1 = o_2 \\
\text{Mapping}(m_1) \land \text{Mapping}(m_2) & \rightarrow \left( (\forall \text{or ObjectRole}(\text{or}) \quad \forall o \text{ Object}(o) \ (\text{Maps}(m_1, \text{or}, o) \leftrightarrow \text{Maps}(m_2, \text{or}, o))) \rightarrow m_1 = m_2 \right)
\end{align*}
\]
StructuralConditionExObject(bct, or₁, ot₁) ∧ ... ∧ StructuralConditionExObject(bct, orₙ, otₙ) ∧ IsOfType(ο₁, ot₁) ∧ ... ∧ IsOfType(οₙ, otₙ) → ∃m Mapping(m) ∧ Maps(m, or₁, o₁) ∧ ... ∧ Maps(m, orₙ, oₙ)

Now, we can determine whether the structural conditions are satisfied for a given behavior constituent type, bct, and a given mapping, m, of its roles:

SC(bct, m) ↔
(StructuralConditionExObject(bct, or, ot) → ∃o Object(o) ∧ IsOfType(o, ot) ∧ Maps(m, or, o)) ∧
(StructuralConditionExRelationTuplen(bct, rel, or₁, ..., orn) → ∃rt ∃o₁ ... ∃on RelationTuple(rt) ∧ RelationTuplen(rt, rel, o₁, ..., on) ∧ Maps(m, or₁, o₁) ∧ ... ∧ Maps(m, orn, on))

Remarks: When using symmetric relations in the structural conditions, there can be two separate matches for each "direction" of the relation. Depending on a potential symmetry of the behavior constituent type, it would be possible to mark it for occurring only once, but we have decided against this option. For example, a turbulent mixing process involving neighboring water compartments will typically be designed symmetrically - both in the conditions and the effects. But we argue that it is always possible to modify the behavior constituent to represent only "half" of the mixing process, e.g. by calculating only half the exchange rate, if numerical values are significant.

An important restriction for structural conditions as described here is that one can only express "positive" conditions, i.e. in terms of objects and relations that have to exist in order to make the condition true. We have not provided for modeling primitives to require the absence of objects or relations and, hence, it is not possible to use conditions of the form "... and there is no object of type X in relation Y with Z". This is primarily a problem of elegantly specifying "negative" conditions - we have found that our algorithmic design could handle the semantics with only minor modifications. See the respective section in the specification of $G^2DE$ (section 7.2) and also the discussion of negative structural effects ("destructive processes" in section 5.3.3.4 below).

When aiming at a faithful reconstruction of the classical modeling constructs of the component-oriented case, one will also want to use the mechanism of shared "terminals". Thus, there would be a pipe model relating quantities associated with terminals that are shared between the pipe and adjacent components, respectively. The structural conditions would look like the diagram in figure 5.10:

![Figure 5.10: Structural conditions of a Pipe component model](image)

Note that a component model should not introduce new structural elements. All relevant quantities, such as substance concentrations, will have to be known in advance and be coded into the PipeTerminal object type (by quantity associations, in our terminology). But compare section 8.3.2 for an example on how structures composed of components and terminals can be enriched by structural aggregations.
5.3.3.2 Quantity Conditions

The specification of quantity conditions (QC) also makes use of the object roles defined by the structural conditions. In connection with the quantity roles defined in the quantity associations from the Ontology section, this enables the modeler to identify quantities associated with the matched configuration - and to specify constraints to be fulfilled for a set of quantities.

Syntax: A quantity specification referring to a bound object (later in the discussion of the quantity effects, we will introduce another kind of quantity specifications) is a pair consisting of an object role - w. r. t. the behavior constituent - and a quantity role - w. r. t. the object bound to the role. Formally,

\[ \text{QuantitySpecificationObjectRole}(qs, or, qr) \]

with \( qr \) designating a quantity role of the object bound to the object role \( or \), of the behavior constituent type.

The quantity conditions themselves are constructed using pre-defined constraint types. A given constraint type is supplied with an ordered set of quantity specifications matching its arity, and the resulting constraint template is marked as being part of the quantity conditions of a specific behavior constituent type:

\[ \text{ConstraintType}(ct) \]
\[ \text{ConstraintTemplate}_n(ctpl, ct, qs_1, ..., qs_n) \]
\[ \text{QuantityCondition}(bct, ctpl) \]

Remarks: As mentioned before, our approach is open to different constraint solvers using different constraint type sets, that are potentially even extensible with user-defined constraints. This is the case with the constraint system used in the prototypical G'DE implementation (see section 7.3), which allows the definition of constraints using truth tables, and even a hierarchical composition of arbitrary constraints over finite domains.

Using constraints as conditions relies on the basic query, whether the constraint is fulfilled for given quantity values. Of course, advanced reasoning as in a diagnostic task will make use of the information contained in the model in more than one way, e. g. by determining the values that must be present, if the behavior constituent is concluded to occur. We will discuss this in the next section, as well.

We have not provided for logical connectives to build complex conditions from simpler ones, except that the quantity conditions as a set form a conjunction. Nor have we allowed for the introduction of auxiliary variables to simplify complex calculations, e. g. to compare the difference of two quantities with a given threshold, in analogy to the quantity effects (see below). Both of these extensions would add convenience, but there is no principled limitation if the constraint system allows for the construction of arbitrary constraints.

Example: In figure 5.11, the quantity conditions of IronRedissolving are shown: The pH of the water layer (bound to) aLayer has to be below a given threshold (constraint type LESS-TAN-x) and the concentration of the solid iron in the sediment has to be positive. Naming a concentration quantity in association with the object role IronSediment in the structural conditions is an example of a quantity specification. Together with this quantity specification, the predefined constraint type POS forms a constraint template.
Semantics: A mapping, as used in the definition of structural conditions, induces a binding of quantities to quantity specifications. We use the predicate Binds for describing this:

\[
\text{Binds}(m, qs, qu) \leftrightarrow \exists qr \exists or \exists o \text{QuantitySpecificationObjectRole}(qs, or, qr) \land \text{Object}(o) \land \text{Maps}(m, or, o) \land \text{HasQuantityRole}(qu, o, qr)
\]

This allows us to state the conditions to be fulfilled:

\[
\text{QC}(bct, m) \leftrightarrow \text{QuantityCondition}(bct, ctpl) \land \text{ConstraintTemplate}(ctpl, ct, qs_1, ..., qs_n) \land \text{Binds}(m, qs_1, qu_1) \land ... \land \text{Binds}(m, qs_n, qu_n) \land \text{HasValue}(qu_1, v_1) \land ... \land \text{HasValue}(qu_n, v_n) \rightarrow \text{ConstraintSatisfied}(ct, v_1, ..., v_n)
\]

The predicate ConstraintSatisfied is special in that it is defined externally. For the sake of simplicity, we assume that we can check for any vector of values whether they satisfy a given constraint type.

5.3.3.3 The Fundamental Laws of Behavior Constituents

We have stated earlier, that for each behavior constituent type, there is a mapping from the set of object roles (from the structural conditions) to all objects that could possibly take these roles. Now, we can select from these mappings the configurations that will actually trigger a behavior constituent - and its effects. The core of the semantics is expressed in two occurrence rules, the first of which is:

First occurrence rule:

\[
\text{OR}_1 \quad \text{SC}(bct, m) \land \text{QC}(bct, m) \rightarrow \exists m' \exists bc \quad \text{Extends}(m', m, bc) \land \text{BehaviorConstituent}(bc, bct, m')
\]

\[
\text{BehaviorConstituent}(bc, bct, m) \rightarrow \text{SC}(bct, m) \land \text{QC}(bct, m)
\]

This rule ensures that a behavior constituent will come into existence if and only if its conditions are met. The effects are defined using an extension of the mapping matching the structural conditions. The extended mapping, \(m'\), will correspond to the original mapping, \(m\), in all object roles from the structural conditions of the behavior constituent type - but will also encompass newly introduced elements from the structural effects (see below).
An extension of a mapping is specified as

\[
\text{Extends}(m, m', \text{bct}) \rightarrow \text{Mapping}(m) \land \text{Mapping}(m') \land \text{BehaviorConstituentType}(\text{bct})
\]

\[
\text{Extends}(m, m', \text{bct}) \leftrightarrow \left( \forall \text{ot} \left( \forall \text{or} \left( \text{StructuralConditionExObject}(\text{bct}, \text{or}, \text{ot}) \rightarrow (\text{Maps}(m, \text{or}, \text{o}) \rightarrow \text{Maps}(m', \text{or}, \text{o})) \right) \right) \right)
\]

A behavior constituent is confined by the following definitions:

\[
\text{BehaviorConstituent}(\text{bc}, \text{bct}, m) \rightarrow \text{BehaviorConstituentType}(\text{bct}) \land \text{Mapping}(m)
\]

\[
\text{BehaviorConstituent}(\text{bc}_1, \text{bct}_1, m_1) \land \text{BehaviorConstituent}(\text{bc}_2, \text{bct}_2, m_2) \rightarrow
\begin{align*}
\text{bc}_1 = \text{bc}_2 & \leftrightarrow \text{bc}_1 = \text{bc}_2 \land m_1 = m_2
\end{align*}
\]

The structural and quantity effects result from the existence of a behavior constituent - this constitutes the

**Second occurrence rule:**

\[
\text{OR}_2 \quad \text{BehaviorConstituent}(\text{bc}, \text{bct}, m) \rightarrow \text{SE}(\text{bc}) \land \text{QE}(\text{bc})
\]

Note that the (extended) mapping, \(m\), will be partly defined by \(\text{SE}(\text{bc})\), see below.

### 5.3.3.4 Structural Effects

As structural effects (SE) of a behavior constituent, new objects and relations can be created. This can be used to model creative processes, e. g. a chemical reaction forming a new compound from other substances. Note that we have not provided for destructive processes that can make existing objects disappear.

The reason for this lies in the overall approach using a first-order logical theory as semantics. Especially, structural effects could be described better in a temporal logic, using concepts of action and change. However, our snapshot approach has some advantages for consistency-based diagnosis and so we define the semantics of a "positive" structural effect as something like ". . . if the behavior constituent occurs, there is always also an object / a relation . . . " . This basically locates the effect in the same time instant together with the conditions of the behavior constituent, thus reducing the reasoning about action and change to finding consistent states. As long as there are only "positive" structural conditions and "positive" structural effects, this does not create any problems. "Negative" conditions or effects, however, require more sophisticated reasoning methods.

**Syntax:** We provide the following primitives for expressing the structural effects of a behavior constituent type \(\text{bct}\):

\[
\text{StructuralEffectExObject}(\text{bct}, \text{or}, \text{ot})
\]

\[
\text{StructuralEffectExRelationTuple}(\text{bct}, \text{rel}, \text{or}_1, ..., \text{or}_n)
\]

This is intended to also bind new object roles, \(\text{or}\), to the newly created objects (of object type, \(\text{ot}\)).

**Example:** See figure 5.12 for the corresponding graphical representation. The structural effects of \text{IronRedissolving} are defined as introducing dissolved iron (\text{IronWater}) into the water layer \text{aLayer}. Both the object instance and the relation tuple (relation dissolved-in) are part of the effects specification:
**Semantics:** As mentioned before, SE(bc) will serve both to ensure the existence of all structural elements specified as effects and to include the binding of their roles into the extended mapping associated with the behavior constituent, bc:

\[
\text{SE(bc)} \leftrightarrow \\
\text{BehaviorConstituent(bc, bct, m)} \rightarrow \\
(\text{StructuralEffectExObject(bct, or, ot)} \rightarrow \\
\exists o \text{ Object(o) } \land \text{IsOfType(o, ot)} \land \text{Maps(m, or, o)}) \land \\
(\text{StructuralEffectExRelationTuple}(bct, rel, or1, ..., orn) \rightarrow \\
\exists rt \exists o1 ... \exists on \text{ Object(oi)} \land \text{Maps(m, ori, oi)} \land \text{RelationTuple}(rt, rel, o1, ..., on))
\]

**Remarks:** We have to take care about the introduction of elements that "are already there". One can determine whether an object template (as defined by the object role in the structural conditions) refers to an existing object by using the relation properties of "uniqueness" or "functionality" (see section 5.3.1.2 above). For example in a IronRedissolving behavior constituent, the reasoning system will be required to check the unique relation dissolved-in for iron present in the water layer bound to aLayer, and if there already is an iron object, bind it to the object role IronWater, rather than create a new one. Its concentration quantity will be used in the quantity effects (see below) for manipulating the(!) iron concentration in aLayer.

Usage of relation properties also requires care: If "complete" relations are involved, the set of existing and created objects and relation tuples in a behavior constituent type must be closed w. r. t. these relations. This means, if an object (e. g. a Substance) is introduced, for which a relation (e. g. dissolved-in) is defined with the completeness property, than there also has to be another object (a LiquidLocator) in this particular relation with the first one. Now, if a modeler does not specify this relation (we did in the example in figure 5.12), we are confronted with an ambiguity. This creates undesirable problems in reasoning, since all existing objects would have to be checked whether they could be unified with the required object (the LiquidLocator) and a potentially implicitly created object could give rise to other problems of the same category (maybe there has to be a master compartment for each compartment etc.). So, in our implementation we decided to require the user to close behavior constituent type definitions w. r. t. complete relations - also see the unification rules in appendix A.
5.3.3.5  Quantity Effects

Finally, the quantity effects (QE) of a behavior constituent are what actually causes changes in quantity values. They can be specified similar to quantity conditions, by employing quantity specifications and constraint templates - but, of course, the constraint is enforced rather than tested. In contrast to the quantity conditions, quantity effects can rely on object roles both from the structural conditions and from the newly created objects in the structural effects. Additionally, a behavior constituent can introduce an arbitrary number of internal variables, called behavior constituent quantities. These can be used for more complex calculations, e.g. to first determine a process "rate" by using quantities of the bound objects and then to use this process rate in the specification of the actual effects.

Furthermore, there is another important construct for describing quantity effects besides constraints, namely influences. We use influences slightly different to QPT, but the basic meaning is the same: providing means for specifying partial effects, that can be composed with others. Again, our approach is very open in the sense that the set of influences is not fixed. Rather, we expect different kinds of influences (influence types) to be supplied - and their semantics to be defined for complete models, i.e. we must be able to determine a constraint for the value of the influenced quantity if all influences are known - for a more detailed discussion of influence resolution see section 5.3.3.6 below.

Syntax: Constraints as quantity effects are specified like this:

```
QuantityEffectConstraint(bct, ctpl)
```

where `bct` is a behavior constituent type and `ctpl` is a constraint template.

For defining behavior constituent quantities for the behavior constituent type `bct`, the following predicate is used:

```
QuantityAssociationBCType(bct, qr, qt)
```

In analogy to `QuantityAssociationObjectType(ot, qr, qt)` (see section 5.3.2 above), `qr` designates a quantity role and `qt` the respective quantity type. For referring to these quantities in quantity effects, we employ quantity specifications of the type

```
QuantitySpecificationBCType(qs, qr)
```

using only a quantity role, `qr`, to be resolved within the given behavior constituent instance.

Influence types are provided as primitives:

```
InfluenceType(it)
```

and, in analogy to constraint templates, we construct influence templates and declare them as quantity effects:

```
InfluenceTemplate(itpl, it, qs1, qs2)
QuantityEffectInfluence(bct, itpl)
```

The quantity specifications, `qs1` and `qs2`, are given in the order source - destination.

Remarks: Enforcing constraints generally leads to restrictions on quantity values, e.g. a liquid level in a tank is constrained to be non-negative. In the extreme - but quite common - case, this confines quantities to a single value, thus providing a mechanism for calculations. Examples are mathematical operations like addition and subtraction, but note that for qualitative values these operations are not necessarily deterministic.
**Example:** Usually, the quantity effects comprise a decisive part of a behavior constituent type definition, as is the case with IronRedissolving (see figure 5.13). A behavior constituent quantity rate (of quantity type Sign) is used to define the effects of the transport. It increases with the source iron concentration and decreases with pH (constraint DIV for a division). This is, of course, an abstracted description, since coefficients relating the flow and the concentrations are neglected and the quantity type would usually reflect at least some significant distinctions other than zero. In dependence of the rate, the concentration of IronSediment is influenced negatively and the concentration of IronWater is influenced positively.

![Figure 5.13: Quantity effects of IronRedissolving](image)

**Semantics:** First, we have to expand the induced binding of quantity specifications to actual quantities in order to cover the newly introduced behavior constituent quantities (see the definition of QE below for their creation):

\[
\text{Binds2}(bc, qs, qu) \leftrightarrow \\
(\exists m \exists bc \text{BehaviorConstituent}(bc, bct, m)) \land \\
((\exists qr \exists or \text{QuantitySpecificationObjectRole}(qs, or, qr) \land \\
\text{Binds}(m, qs, qu)) \lor \\
(\exists qr \text{QuantitySpecificationBCType}(qs, qr) \land \\
\text{HasQuantityRoleBC}(qu, bc, qr))
\]

Using this, we can define:

\[
\text{QE}(bc) \leftrightarrow \\
\text{BehaviorConstituent}(bc, bct, m) \rightarrow \\
(\text{QuantityAssociationBCType}(bc, qr, qt) \rightarrow \exists qu \text{HasQuantityRoleBC}(qu, bc, qr)) \land \\
(\text{QuantityEffectConstraint}(bc, ctpl) \land \text{ConstraintTemplate}(ctpl, ct, qs_1, ..., qs_n) \land \\
\text{Binds2}(bc, qs_1, qu_1) \land ... \land \text{Binds2}(bc, qs_n, qu_n) \land \\
\text{HasValue}(qu_1, v_1) \land ... \land \text{HasValue}(qu_n, v_n) \rightarrow \\
\text{ConstraintSatisfied}(ct, v_1, ..., v_n)) \land \\
(\text{QuantityEffectInfluence}(bc, itpl) \land \text{InfluenceTemplate}(itpl, it, qs_1, qs_2) \land \\
\text{Binds2}(bc, qs_1, qu_1) \land \text{Binds2}(bc, qs_2, qu_2) \rightarrow \\
\text{Influence}(it, qu_1, qu_2))
\]
For the newly introduced behavior constituent quantities, we require some laws of uniqueness, similar to the association of quantities with objects:

\[
\text{HasQuantityRoleBC}(qu, bc, qr) \rightarrow \text{Quantity}(qu) \land \text{QuantityRole}(qr) \land \\
(\exists bct \exists m \, \text{BehaviorConstituent}(bc, bct, m))
\]

\[
\text{HasQuantityRoleBC}(qu_1, bc_1, qr_1) \land \text{HasQuantityRoleBC}(qu_2, bc_2, qr_2) \rightarrow \\
(qu_1 = qu_2 \leftrightarrow bc_1 = bc_2 \land qr_1 = qr_2)
\]

The handling of influences will be the topic of the next section.

5.3.3.6 Influence Resolution

Influences are special, since they provide a means of partial and combinable specification of behavior, as influences from different behavior constituents can target the same quantity. The semantics of influences on the other hand are defined for the complete set of influences on a given quantity. This introduces an important non-monotonic element into the theory: we can collect all influences from a set of behavior constituents believed to be complete, and then calculate their effects, only to find that they determine the occurrence of further behavior constituents creating additional influences, so that we might have to retract predictions and conclusions. Of course, this shows that one cannot proceed in this naive manner. For our implementation, we will use a sophisticated combination of a transformation into monotonic composition steps and assumption-based reasoning - see section 7.1 for a description.

But first, we define an abstract framework for the semantics of influences: We assume that for each set of influence types that can combine (i.e. can share the same target quantity), a (unique) summary constraint type is provided:

\[
\forall it_1 \ldots \forall it_n \, \text{InfluenceType}(it_1) \land \ldots \land \text{InfluenceType}(it_n) \rightarrow \\
\exists ct \, \text{ConstraintType}(ct) \land \text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \\
\text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \land \text{SummaryConstraintType}(ct', it_1, \ldots, it_n) \rightarrow ct = ct'
\]

In all cases considered here, there are influences of at most two different types for a single quantity - and the summary constraint type is expected to be commutative within each type, i.e. the order of values combined for each type does not matter.

\[
\text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \land it_i = it_j \rightarrow \\
(\text{ConstraintSatisfied}(ct, v, v_1, \ldots, v_i, \ldots, v_n) \leftrightarrow \\
\text{ConstraintSatisfied}(ct, v, v_1, \ldots, v_j, \ldots, v_n))
\]

As with constraints, the framework is open for the addition of user-defined influence types by specifying the associated summary constraint. However, we find that a set of pre-defined influence types is sufficient for most tasks.

Standard types defined are additive and multiplicative influences. For additive combination, we have the types \text{fct-pos} and \text{fct-neg} (see the example in figure 5.13 for their usage and figure 5.14 below for an overview of pre-defined influence types). The summary constraint type calculates the sum. For influence types \(it_1, \ldots, it_n \in \{\text{fct-pos}, \text{fct-neg}\}\) and the helper function

\[
f_\cdot : \{\text{fct-pos}, \text{fct-neg}\} \rightarrow \{+1, -1\}
\]

\[
f_\cdot(fct-pos) = +1, f_\cdot(fct-neg) = -1
\]
we have
\[
\text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \rightarrow \\
(\text{ConstraintSatisfied}(ct, v, v_1, \ldots, v_n) \leftrightarrow (v = f_+(it_1) \cdot v_1 + \ldots + f_+(it_n) \cdot v_n)
\]

From a tradition of pure influence diagrams (see [Heller/Struss 1996]), we have kept compatibility with multiplicative combination, which is seldom used. Usually, a multiplication constraint can be directly specified instead, since multiplicative influences are rarely used for partial behavior specifications. Still, influences of the types fct-mul and fct-div can be resolved into a product constraint. Employing the helper function
\[
f_+: \{\text{fct-mul, fct-div}\} \rightarrow \{+1, -1\}
\]
\[
f_+(\text{fct-mul}) = +1, f_+(\text{fct-div}) = -1
\]
the general scheme is
\[
\text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \rightarrow \\
(\text{ConstraintSatisfied}(ct, v, v_1, \ldots, v_n) \leftrightarrow (v = v_1 f_+(it_1) \cdot \ldots \cdot v_n f_+(it_n))
\]

Now we can define the influence resolution for the complete set of influences on a given quantity, \(qu\),
\[
\text{Influence}(it_1, qu_1, qu) \land \ldots \land \text{Influence}(it_n, qu_n, qu)
\]
as the following law for the combination of the respective values:
\[
\text{SummaryConstraintType}(ct, it_1, \ldots, it_n) \land \\
\text{HasValue}(qu, v) \land \text{HasValue}(qu_1, v_1) \land \ldots \land \text{HasValue}(qu_n, v_n) \rightarrow \\
\text{ConstraintSatisfied}(ct, v, v_1, \ldots, v_n)
\]

The important part is the completeness of the set of influences, which is intimately connected to the issue of the closure of the model, described in section 5.3.5. We will see that the set of influences is closed with reference to the set of behavior constituents occurring, which in turn depends on the set of structural elements. Assuming a given set of elements closed will then result in a closed set of influence - and the exploitation and challenge of this closure assumption is a core part of calculating valid solutions (see section 7.2.2).

But there is another set of influence types to be mentioned, namely "integrative" ones, which are dependent on the inclusion of derivatives in the model. Although we haven't discussed it yet, there is the option of automatically generating a derivative quantity for each quantity present in the model. See section 7.2.5.3 for automated model enhancements. This is the only place in the modeling language, where we provide an explicit reference to these enhancements.

The idea is that an integrative influence (types int-pos, int-neg) works in analogy to an additive one (types fct-pos, fct-neg) only that the summary constraint is created for the derivative of the target quantity. See figure 5.14 for the pre-defined types of influences that can be used in modeling quantity effects.

![Figure 5.14: Pre-defined influence types](image-url)
A final issue to be discussed here is the combination of influences and constraints for a quantity. We assume that both the influence combination (summary constraint) and all other constraints directly involving the quantity are valid in an unrestricted fashion. This requires the modeler to exercise care in the use of constraints, as they have to be checked for general validity. For instance, the common modeling technique of supplying a constraint for the equilibrium equation of a fast process can quickly lead to unexpected discrepancies, if additional processes are discovered that influence the equilibrium. Note that this can be exactly the intended effect in other cases, where the user wants to diagnose deviations from equilibrium.

5.3.4 Basic Axioms

As mentioned before, the invariable axioms relating the modeler’s statements are collected into a separate section of the system description. All important laws have been presented directly with the description of the modeling primitives, but it is beyond the scope of this thesis to achieve axiomatic completeness. The reason for including these "semantic" definitions in the model is to make the approach comparable to the classical theories, where the system description, SD, contains all needed first-order sentences.

The absence of axioms for temporal reasoning (defining continuity of quantity values or default object persistence) is a consequence of the state-based approach to diagnosis proposed here. It remains to be examined how the approach can be generalized to temporal diagnosis, especially w. r. t. the closed-world assumptions used in consistency check. For now, we will exclude these aspects from the formalization.

5.3.5 Closure of the Domain Theory

One more set of axioms of central importance has to be discussed here, namely the closure of the domain theory. Firstly, the domain theory is closed in a static sense, i. e. enumerations, as of object types, relations, quantity types, or behavior constituent types are taken as complete - it is not part of any valid solution to "conclude" a new behavior constituent type to make a given situation consistent. Formally, this can be expressed as a set of static closure axioms of the form

\[
\text{ObjectType}(ot) \rightarrow ot = ot_1 \lor \ldots \lor ot = ot_n
\]
\[
\text{Relation}(rel) \rightarrow rel = rel_1 \lor \ldots \lor rel = rel_m
\]
\[
\text{BehaviorConstituentType}(bct) \rightarrow bct = bct_1 \lor \ldots \lor bct = bct_l
\]

for given finite sets of object types (\(ot_1\) through \(ot_n\)), relations (\(rel_1\) through \(rel_m\)), or behavior constituent types (\(bct_1\) through \(bct_l\)) and so on. Similarly, we have to close the definition of object supertypes and the conditions and effects of a behavior constituent type. This can, for instance, be written as

\[
\text{IsA}(ot_i, ot') \rightarrow ot' = ot_i \lor \ldots \lor ot' = ot_n
\]
\[
\text{StructuralConditionExObject}(bct, or, ot) \rightarrow (or = or_1 \land ot = ot_1) \lor \ldots \lor (or = or_m \land ot = ot_m)
\]
\[
\text{StructuralEffectExObject}(bct, or, ot) \rightarrow (or = or_1 \land ot = ot_1) \lor \ldots \lor (or = or_l \land ot = ot_l)
\]

The following is the list of predicates closed in this fashion:

- ObjectType
- IsA
- Relation
- QuantityType
• QuantityAssociationObjectType
• BehaviorConstituentType
• QuantityAssociationBCType
• StructuralConditionExObject
• StructuralConditionExRelationTuple
• QuantityCondition
• StructuralEffectExObject
• StructuralEffectExRelationTuple
• QuantityEffectConstraint
• QuantityEffectInfluence

Of course, these closure axioms are not an invariable part of the Basic Axioms section, since their form depends on the user-defined part of the model. However, for the sake of simplicity, we will not designate a special section for them - one can view them as an implicit part of the user-defined sections.

Secondly, there are closure axioms for predicates the extension of which is not known from the beginning. Most importantly, an influence can be concluded from the quantity effects of an occurring behavior constituent, but we want to exclude the existence of influences that are not supported by a behavior constituent instance. This can be expressed in the form of the

**Influence closure axiom:**

\[(IC) \quad \text{Influence}(it, qu_1, qu_2) \rightarrow \\
(\exists bc \exists bct \exists m \text{BehaviorConstituent}(bc, bct, m) \land \\
\exists itpl \exists qs_1 \exists qs_2 \text{InfluenceTemplate}(itpl, it, qs_1, qs_2) \land \text{QuantityEffectInfluence}(bct, itpl) \land \\
\text{Binds2}(bc, qs_1, qu_1) \land \text{Binds2}(bc, qs_2, qu_2))\]

We would like to point out the similarity to the "sole mechanism assumption" used by Forbus ([Forbus 1984]). The causal perspective taken in QPT dictates the formulation that no value change can occur except as an effect of a process. Our approach is consistency-based and, so far, neglects the temporal aspect. Therefore, the focus is not on value changes, but rather on the laws constraining a value at a given point in time, but see section 7.2.2.1 for the restriction of derivatives, which comes very close to the original assumption from QPT.

Analogously, one could close the extensions of constraints, by requiring support by an occurring behavior constituent, as well. However, we consider the structural elements and the quantity values as the relevant part of a solution - and additional constraints cannot create additional solutions in terms of structural elements and values, but merely can restrict them. Hence, the closure of constraints is not necessary for defining valid solutions.

As a remark, the situation description (see next section) is closed in a different fashion - minimal extensions of the predicates used there are defined as valid solutions. A formal treatment of minimization in the framework of default logic is presented in section 6.2.
5.3.6 Situation Description

Diagnosis starts with what is known of the situation. This includes an initial system structure and observations of quantity values. In our modeling language, the basic elements are object instances and relation "tuples", as well as assignments of quantity values. However, this situation description is open to change (by the diagnostic task) in two important respects:

On the one hand, it is usually incomplete, i.e. only a partial description of the situation as characterized by objects, relations and quantity values. It is impractical and very often impossible to measure or observe all of the quantities in the system model, they can only be predicted from other sources of knowledge. But also unexpected objects and relations might be present, which is left for the diagnosis procedure to discover. So in this respect, the situation description is incomplete and open to completion.

On the other hand, the user might want to describe an assumed or "default" situation and specify observations with limited confidence. We allow to supply structural elements (objects and relations) and also quantity value assignments with user-defined assumptions. This distinguishes these specifications from facts. A user-defined assumption is identified by a constant that can be used in more than one location, e.g. for supplying multiple values with the assumption that the common measurement unit providing them is working correctly. The diagnostic system is able to retract assumptions, if they contradict facts or other assumptions. The reasoning process determines which assumptions to retract by following the same procedure as with mode assumptions in standard component-oriented diagnosis (see section 3) - actually, our implementation uses a standard diagnosis system for this task. Sophisticated algorithms for finding the most likely sets of assumptions to retract are available. Thus, part of the situation description can be made open to retraction, as well.

Structural elements are defined by identifying a constant, o, as an object instance and providing an object type, ot:

\[
\text{Object}(o) \\
\text{IsOfType}(o, ot)
\]

and a relation tuple, rt, is characterized by a relation, rel, from the Ontology section with its object parameters, o₁ through oₙ:

\[
\text{RelationTuple}(rt) \\
\text{RelationTuple}(rt, rel, o₁, ..., oₙ)
\]

The main axioms related to these predicates have already been described in section 5.3.1. As a convenient shortcut, as we often treat structural elements alike in formal proofs, we introduce the predicate

\[
\text{Element}(e) \leftrightarrow \text{Object}(e) \lor \text{RelationTuple}(e)
\]

A quantity, qu, is identified using an object instance, o, and a quantity role, qr (see also section 5.3.2 for a characterization of the following predicate):

\[
\text{HasQuantityRole}(qu, o, qr)
\]

with qu being a quantity, which can be assigned a value, v:

\[
\text{HasValue}(qu, v)
\]
Optionally, these expressions are provided with an antecedent consisting of a conjunction of assumptions. Assumptions are introduced as

\[
\text{Assumption}(assm)
\]

and they are used with the predicate \text{Holds} in expressions of the following form:

\[
\begin{align*}
\text{Holds}(assm_1) \land \ldots \land \text{Holds}(assm_n) & \rightarrow \text{Object}(o) \\
\text{Holds}(assm_1) \land \ldots \land \text{Holds}(assm_n) & \rightarrow \text{RelationTuple}(rt) \\
\text{Holds}(assm_1) \land \ldots \land \text{Holds}(assm_n) & \rightarrow \text{HasValue}(qu, v)
\end{align*}
\]

A remark on the specification of goals for diagnosis and therapy recognition: Only quantity value assignments will be used for this purpose. We neither allow the specification of a "target system structure" directly (we argue that a structure will do, if the observable goals are achieved), nor do we provide complex constructs like defining constraints or temporal trajectories for quantity values.
6 Characterization of Tasks and Solutions

Given a model of the kind described in the last section, what kind of conclusions can be drawn? We find that the domain specific part of the model can be described as complex rules of the kind "if X is the case, then also Y is the case". Of course, from these rules alone we cannot derive anything to actually be the case. But provided with a (partial) situation description as a starting point, the model can be used for predicting variable values and even the presence of objects or relations. In prediction, the rules from the domain theory are mainly used in the "forward" direction, i.e. when establishing that the conditions of a behavior constituent are satisfied, its effects are also known to occur.

But there is a crucial exception, namely the combination of influences, which gives rise to a very interesting kind of conclusion. Determining the effects of a number of influences on a target quantity requires knowledge about the complete set of influences, thereby assuming that all other influences are absent. The interesting point is that, for instance, a set of influences that taken by itself would make a quantity increase, but that is in contradiction to the observation of the quantity decreasing, can be deduced to be incomplete! Additionally, since we consider the domain theory to be closed, it is possible to deduce the set of potentially occurring behavior constituent instances capable of bringing about the required additional influences on the quantity - and the conditions that have to be fulfilled for them to do so. At least one of these conditions has to be true, if we adhere to the closure principles formalized in section 5.3.5. Thus, the logical theory allows for an object existence to be implied by its observed effect (via a behavior constituent instance) on a quantity.

This way of using the domain theory rules in the "backward" direction proves to be a very powerful tool for reasoning about a situation, a task we call "situation assessment". Potentially unexpected "deep" causes of the currently observed behavior can be discovered by logical deduction from the observations and the domain theory. But this direction of reasoning can also be used for a quite different task, namely finding out what might make the system behave as desired. By specifying a desired situation ("goals" in our terminology), the model can be used to derive what can be done to bring it about! This is the foundation for model-based therapy recognition. This is a necessary differentiation of "the" single diagnostic task as defined for the classical component-oriented approach.

Note that all kinds of reasoning mentioned above are basically completion of a partial situation description. Complementing it, we provide methods for the retraction of (assumed) structural or behavioral specifications. Thus, in situation assessment, both unexpected objects or other conditions can be discovered and evidence for the absence of specified objects or conditions can be collected. In terms of model revision, we speak of correcting errors of omission and errors of commission. Transferred to therapy recognition, this corresponds to finding actions to start and phenomena to stop - or rather elements to add and to remove. It is the seamless integration of both techniques that constitutes the power of the presented approach.

In the following, we present a formal treatment of the tasks of situation assessment and therapy recognition (section 6.1) including a discussion of the control of these tasks through the specification of revisables (section 6.1.3). Then, we formally characterize the acceptable solutions to these tasks using a non-monotonic logic framework (section 6.2). Of course, there are criteria, mainly of parsimony, for selecting from the possibly infinite number of solution candidates, so we conclude this chapter with some remarks on candidate ranking (section 6.3).
6.1 Differentiation of Diagnostic Tasks

For component-oriented consistency-based diagnosis, the usual perspective is that there is a single well-defined diagnostic task. In section 3, we have analyzed the fundamental assumptions underlying this view - and we find that few hold equally for the generalized case. In particular, the standard approach can rely solely on the observations as opposed to the system structure. This is because the system structure implies the desired or correct working of the system "by design". When reasoning about a wider range of systems, many of which are not designed by human engineers for a specific purpose, this view proves to be too narrow. E. g. an observed high concentration of dissolved iron is, of course, in perfect conformance with the laws of nature - the contradiction is with human goals external to the system. Reasoning about this discrepancy, its causes and possible remedies for it, requires an explicit representation of these goals.

We call such as explicit representation GOALS. In the component-oriented case, the design of the system is expected to ensure

\[ SD \models \text{GOALS} \]

or rather, written with mode assignments:

\[ SD \cup \{\text{ok}(C_i) : C_i \in \text{COMPS}\} \models \text{GOALS} \]

In the following, we examine the differentiation of diagnostic tasks, when this basic assumption is challenged. Section 6.1.1 defines the task of situation assessment, as opposed to therapy recognition (section 6.1.2). In section 6.1.3 the fundamental control of these tasks by the specification of revisables is discussed.

6.1.1 Situation Assessment

If there is no system design with designated \text{ok} modes ensuring the desired behavior, explicit knowledge about the goals in the form of a specification of the allowed range of behaviors is required for identifying (the causes of) deviations. Then, we look at the diagnostic procedure from a new perspective and find a differentiation between two tasks that are identical for the classical approach. The first task is to answer the question "What's going on?", and observations are used to answer it. We call this task "Situation Assessment" and define it as finding a revised system description, \( SD' \), that is consistent with the observations:

\[
\text{Situation Assessment: } SD \cup \text{OBS} \models \perp \Rightarrow SD' \cup \text{OBS} \not\models \perp
\]

The result of this step, \( SD' \), contains a structural description and quantity value assignments (which is more general than mode assignments, as in the classical case) and provides us with a hypothesis about the state of the system under consideration. This will be used as the basis for determining "what can be done" (see below). Note that the two stage procedure proposed here relies crucially on \( SD' \) to have strong implications. While it is possible to achieve the consistency required in the definition of situation assessment with a very weak system description, this will not help much in answering further questions. Ideally, one will require an abductive solution, i. e.

\[
\text{(Abductive situation assessment): } SD' \models \text{OBS}
\]

As a remark, [Console/Torasso 1991] shows how model closures puts the generally weaker consistency-based approach closer to the abductive one.
6.1.2 Therapy Recognition

Classically, components are what can fail and what can be repaired (i. e. usually replaced) at the same time. This corresponds to the notion of a "smallest replaceable unit". Thus, in the component-oriented approach the situation assessment result carries all the information needed for repair. This is not necessarily so in the generalized case, since knowing causes or sources of misbehavior might be completely different from knowing what can be done to improve the situation. Replacing parts of an ecosystem is not (always) an option, nor is there usually a way of undoing damage to it by directly removing pollutants. In the introductory example (section 2), the sedimonial iron discovered in situation assessment leaves open the question what to do about its effects. So, we find that therapy recognition is a separate task, which can also be described as a revision of an system description (a result of situation assessment) inconsistent with a criterion into a consistent one:

Therapy Recognition: $SD' \cup GOALS \models \bot \Rightarrow SD'' \cup GOALS \not\models \bot$

Certainly, the task of therapy recognition would also benefit from abductive reasoning, since ultimately we want to ensure the goals and not only be unable to prove a violation. From a practical perspective, a plausible $SD'$ will have to be "committed" before therapy recognition, as there might be multiple possible situation assessments.

An important difference of the two tasks that is not obvious from these formulas lies in the set of revisables. For the former, "assumable" elements, effects and parameter values are eligible. For instance, situation assessment allows the identification of unexpected objects, such as dissolved iron. Usually, not all configurations of new objects or relations are acceptable as explanations "out of nowhere". In narrowing the set of introducible configurations, e. g. by generally excluding dissolved iron, "deeper" causes will have to be found (as we will see below, dissolved iron will then have to be supported as the effect of another behavior constituent) - or the hypothesis will have to be discarded.

For therapy however, we allow only revisions that can be achieved by human actions. In revising the actual system as represented by $SD'$ to one that does not violate the goals anymore, we don't need answers of the general form "it would be better if the system was like X", but rather "you could introduce Y" (if Y is immediately available) or "you could increase Z" (if one has direct influence on Z). In controlling the set of revisables, very different useful solutions can be generated. Note, that as a result of therapy recognition, we would like to enumerate the revisions (as opposed to presenting only the revised system description), so that the single causes or actions can be regarded.

6.1.3 Specification of Revisables

Having distinguished between two different tasks that are mostly identical in the component-oriented context, we can still use the same basic reasoning technique to solve them. In each case, a system description (SD) has to be revised in order to achieve consistency with given observations or goals. For component-oriented diagnosis, the revision consists solely of changes in mode assignments. We allow for a far more general class of revisions in our approach.

In our terminology, the system description consists of a domain theory, which is closed (see section 5.3.5), and a situation description, which is incomplete and potentially contradictory if combined with the domain theory. Using the domain theory, the task is to complete the situation description (this can be prediction of the values of unobserved quantities or additions to the structural specification) and/or to retract parts of it that rely on assumptions that have been refuted. This is to say that the revisable part of
the system description resides entirely within the situation description - here all changes, additions, and removals take place (cf. figure 5.4).

As said above, the integration of both kinds of revisions is a main advantage of the presented approach. This is achieved by employing a generalized form of consistency-based diagnosis. In the following, criteria for the allowable solutions will be derived.

The essential means of controlling the revision is the specification of revisables, i.e. to identify the retractable assumptions (we can speak of "removable" objects and relations as well as "changeable" quantity value assignments) and to characterize what can be added to a situation description. We will not accept arbitrary objects or relations to be added "without cause" as solutions. Rather, we label certain configurations as introducible, meaning that all other structural additions can only appear in a system description if they are named by the user or created as the effect of an occurring behavior constituent. Introducible elements are accepted as ultimate causes without further explanation.

A simple approach is to label certain object types and relations as "introducible" in general - so that a configuration is introducible if and only if it consists solely of objects belonging to an introducible object type and of relation tuples of introducible relations. This is the simple solution being implemented in the current G*DE prototype.

The set of revisables distinguishes the different tasks by assuming a specific meaning: In situation assessment, we search for objects and relations contributing to the currently observed behavior. In the context of therapy recognition, the distinction between introducible and non-introducible elements is even more significant, since it enables us to designate what can be created or otherwise brought about directly by human intervention.

6.2 Formally: Solutions

In contrast to the domain theory, the situation description is not closed, but rather open for completion (see section 6.2.1) - with semantics based on minimization (see section 6.2.2). The integration of completion with assumption retraction (section 6.2.3) is a central advantage of the presented semantics.

6.2.1 Completion of the Situation Description

An important characteristic of the proposed modeling language is that it does not allow for negative propositions in the situation description, i.e. there are no means to explicitly exclude specific objects or relations. The idea behind this is, that everything that is not mentioned is not there - except if it has to be.

There are two ways, an additional object or relation can be identified as necessary and, consequently, be introduced into the situation description: Firstly, the given situation description could trigger a behavior constituent, which will create the additional structural element. Secondly, effects could be observed that cannot be attributed to the situation description and its "forward" completion alone, but rather point at causes not yet considered. This is a core part of the extended diagnostic task, which can be called "backward" completion. We will characterize solutions based on a minimization of such completions (of both kinds) - usually yielding multiple possibilities.
Three sets of predicates are used to distinguish structural elements based on their justification. Objects and relation tuples introduced by the user, i.e. included in the initial situation description, are marked with

\[
\text{KnownObject}(o) \\
\text{KnownRelationTuple}(rt)
\]

and, as a shortcut that will prove valuable in formal deductions:

\[
\text{KnownElement}(e) \leftrightarrow \text{KnownObject}(e) \lor \text{KnownRelationTuple}(rt)
\]

In section 5.3.6, we stated that the user specifies

\[
\text{Object}(o) \\
\text{RelationTuple}(rt)
\]

For each such specification, we add analogous propositions using \(\text{KnownObject} / \text{KnownRelationTuple}\). With respect to the predicates \(\text{KnownObject}\) and \(\text{KnownRelationTuple}\), the situation description is closed in the formal sense, i.e. all objects and relation tuples satisfying these predicates are known from the beginning. Formally, this corresponds to adding a static closure formula of the form

\[
\text{KnownElement}(e) \rightarrow (e = e_1) \lor \ldots \lor (e = e_n)
\]

The "forward" completion is comprised by the objects and relation tuples from structural effects of necessarily occurring behavior constituents, which is expressed by the predicates

\[
\text{EffectObject}(o) \\
\text{EffectRelationTuple}(rt) \\
\text{EffectElement}(e) \leftrightarrow \text{EffectObject}(e) \lor \text{EffectRelationTuple}(e)
\]

The defining laws are

\[
\text{EffectObject}(o) \leftrightarrow \text{Object}(o) \land \\
\exists bc \exists bct \exists m \text{BehaviorConstituent}(bc, bct, m) \land \\
\exists or \exists ot \text{StructuralEffectExObject}(bct, or, ot) \land \text{Maps}(m, or, o)
\]

and, analogously,

\[
\text{EffectRelationTuple}(rt) \leftrightarrow \text{RelationTuple}(rt) \land \\
\exists bc \exists bct \exists m \text{BehaviorConstituent}(bc, bct, m) \land \\
(\text{RelationTuple}_e(rt, rel, o_1, ..., o_n) \rightarrow \\
(\exists or_1 ... \exists or_n \text{StructuralEffectExRelationTuple}_e(bct, rel, or_1, ..., or_n) \land \\
\text{Maps}(m, or_1, o_1) \land \ldots \land \text{Maps}(m, or_n, o_n)))
\]

Note that these definitions are written as an equivalence. This requires the forward completion to be supported by occurring behavior constituents. In most cases, the set of \(\text{EffectElements}\) is closed w.r.t. the set of structural elements being present initially. In special cases, however, the domain theory allows the possibility of "self-supporting loops", in which behavior constituents can rely on elements supported in a loop by the effects of the very same behavior constituents. While we could exclude this by devising stricter and more complex semantics (e.g. by requiring a "creation path" from initial elements to each effect element), it might not always be undesirable.
For the "backward" completion, i.e., structural elements that are deduced to be part of the model by their respective effects, we use the predicates

\[
\text{IntroducedObject}(o) \\
\text{IntroducedRelationTuple}(rt) \\
\text{IntroducedElement}(e) \leftrightarrow \text{IntroducedObject}(e) \lor \text{IntroducedRelationTuple}(e)
\]

As we will see below (section 6.2.2), the minimization semantics will rely on these predicates. But they are also the key to ensure that only structural elements that are specified as "introducible" can occur as the primary causes in the backward completion. In the simple form of generally labeling object types and relations are introducible or non-introducible, this distinction is expressed via:

\[
\text{IntroducibleObjectType}(ot) \\
\text{IntroducibleRelation}(rel)
\]

Both predicates are closed - either by requiring a complete classification from the user or by employing static closure to a set of positive propositions. Now, "introducibility" is stated as a necessary condition for introducing objects or relation tuples:

\[
\text{IntroducedObject}(o) \rightarrow \exists ot \text{ObjectType}(ot) \land \text{IsOfType}(o, ot) \land \text{Introducible}(ot), \\
\text{IntroducedRelationTuple}(rt) \rightarrow \exists rel \exists o_1 ... \exists o_n \text{RelationTuple}(rt, rel, o_1, ..., o_n) \land \\
\text{IntroducibleRelation}(rel)
\]

For perspectives of more general mechanisms to distinguish introducible configurations, see section 9.3.

### 6.2.2 Minimality of Completions

As solutions, we want to allow completions removing inconsistencies arising from the incompleteness of the situation description (for handling of contradicting assumptions see section 6.2.3). These completions could, however, introduce superfluous objects or relation tuples in doing so. An introducible, but entirely unrelated object without discernible effects will neither help nor hinder achieving consistency. Mostly, there is an infinite number of possible completions, very few of which provide any useful hypothesis about the situation - or, depending on the task, what can be done to improve it. Obviously, we want to restrict the set of solutions to the completions that do.

Therefore, we require that only necessary elements are added (in both kinds of completions) to the ones explicitly named by the user. As a first step, we exclude any structural elements that are not contained in any of the defined classes:

\[
\text{Element}(e) \rightarrow \text{KnownElement}(e) \lor \text{EffectElement}(e) \lor \text{IntroducedElement}(e)
\]

Additionally, the respective predicates will be minimized. For the initial situation description this is achieved by static closure (as defined above) and for the forward completion (\text{EffectElement}) this is ensured by the definition of the predicates \text{EffectObject} and \text{EffectRelationTuple}. These classes cannot contain superfluous objects or relation tuples (but see the remark on "self-supporting loops" in section 6.2.1).

The critical part is backward completion, for we cannot apply a simple closure here. Rather, we define semantics based on minimization (of the predicates \text{IntroducedObject} and \text{IntroducedRelationTuple}) w.r.t. set inclusion. A structural addition is minimal, if no proper subset is sufficient, a.k.a. consistent. It is important to note that this cannot be expressed in first-order logic and we will resort to a non-monotonic logic framework, namely default logic ([Reiter 1980], see also section 3.1) to properly...
formalize the intended characteristics of the solution - and later design the calculation of these solutions as a diagnostic task.

Note that set inclusion imposes only a partial order on all possible completions and, thus, usually multiple solutions satisfy minimality w. r. t. set inclusion. These solutions can exhibit largely different cardinalities of introduced structural elements. Ranking and filtering among different (minimal) solutions is discussed briefly in section 6.3.

Using prerequisite-free normal default schemata, we can easily express the minimality preference for fewer introduced structural elements:

\[
\text{(DE) } \neg \text{IntroducedElement}(e) / \neg \text{IntroducedElement}(e)
\]

with the intuitive reading: "if an object or relation tuple is not necessarily present (i. e. its absence is consistent), then it is indeed absent". Adding these schemata to the theory defined by the domain theory, the situation description and all discussed static closures, will yield exactly the minimal completions as extensions - modulo contradictory user-defined assumptions. This will be the topic of the next section.

### 6.2.3 Completion and Retraction

As stated above, revising the initially provided situation description in order to achieve consistency can not only mean completions (as described in the last sections) but also retraction of assumptions. Depending on the usage of assumptions in the situation description (cf. section 5.3.6), this corresponds to the removal of structural elements or the change of quantity values. The integration of these different kinds of revisions will be discussed here.

Using the terminology of model preferences, we can define a second criterion for preferring one logical model of the theory over another, namely that user-defined assumptions have to hold whenever possible. Since we do not state anything about the validity of user-defined assumptions in the theory, this is clearly a criterion completely external to the theory. In analogy to the addition of structural elements, where we require minimality of introductions, here we are looking for maximal consistent subsets of user-defined assumptions.

In default logic, we state that we do not want to retract user-defined assumptions without necessity:

\[
\text{(DA) } \text{Holds(assm)} / \text{Holds(assm)}
\]

In contrast to the default schema for elements (DE) above, which applies to all entities in the universe of discourse, thus effectively limiting the instances of structural elements, the schema (DA) is to be instantiated only for all assumptions, assm, with Assumption(assm).

Of course, both kinds of revisions do interact with each other, since the completions are to be understood w. r. t. the part of the situation description that is not retracted. From a different perspective, the retraction of a single assumption might be traded off against the introduction of multiple objects and relation tuples. Once again, note that retraction and completion also refers to quantity value assignments, which have been mostly neglected in the description so far, since they are governed by different laws: Each quantity is to be assigned exactly one value in a given state. Therefore, completion boils down to finding this assignment, while retraction means changing it, since there has to be another value. It can be difficult for reasoning systems to make use of this knowledge, e. g. by deriving a certain assignment from the fact that all other allowable values lead to inconsistencies, but we defer this discussion to the section on algorithmic design (7.2).
The given characterization of solutions as extensions is concise, but does not provide an effective method for calculating actual extensions for a given theory. This is most obvious with the implicit definition of necessary backward completions, characterized as minimal structural additions needed to remove any inconsistency from influence resolution. The fixed domain theory contains all information defining these solutions, but for actually computing these completions, a sophisticated approach is required. Refer to section 7 for the specification of a transformation of the problem and an algorithm to achieve this.

6.3 Candidate Ranking

By definition, solutions are completions with a minimal set of introduced structural elements and a maximal set of holding assumptions - both criteria are defined in terms of set inclusion, which often leaves us with multiple solutions. This situation is similar to multiple diagnostic candidates in classical component-oriented diagnosis (see section 3), where sophisticated techniques for candidate ranking have been developed. Candidate ranking refers to the selection of preferable candidates from a set of already minimal solutions, with preference criteria usually also based on some kind of parsimony. However, for the theory of process-oriented consistency-based diagnosis presented here, application of a parsimony principle is not nearly as simple as for the component-oriented case. There one could minimize the absolute number of faulty components, as represented by the assignment of a fault mode, and achieve a significant level of plausibility - the failure of multiple components is very rarely more likely than a single fault. This can be extended with a more sophisticated probability model, where components and even their individual fault modes are assigned an a priori probability (see section 3).

But when searching for newly introduced elements or unanticipated interactions, a valid solution found could be more complicated than a single "deeper" cause for the same configuration. Imagine the discovery of a number of pollutants in a water body - all of which are concluded from their observable effects. We could be satisfied with the explanation that these substances somehow appeared in the ecosystem under consideration, ending up with a solution with a high cardinality w. r. t. introduced elements. But imagine further, that it is known that these pollutants are present in a tank of a nearby industrial plant. A single cause, such as the tank leaking into the water body seems a much better solution to the diagnostic task at hand. Similar scenarios bear even more importance in a therapy recognition setting, as multiple actions might be substituted by a single smarter one.

This is also a practical issue, since in many cases, we use a search space and have to stop at a certain acceptable level of depth (the search space is often infinite) - while it is possible that at a deeper level a simpler solution could be found. This suggests, that one should not abandon the search for deeper causes too soon, much less prune all branches of the search tree that reach a high cardinality. However, it not obvious how to control the backward completion process, see section 7.2.4 for a discussion.

In the concise logical definition, we have left it open whether one solution "covers" another one by declaring all of its introduced elements as the effect of another introduced element. Both solutions are treated equally valid from the perspective of our theory. In our prototype, we have basically implemented a "brute force" approach to backward completion, which disregards the size of intermediately "creatable" element configurations (see section 7.2.4). However, the current algorithm stops and reports discovered configurations consisting of introducible structural elements and leaves it to the user to request a further search for deeper causes.
In this section, we present a reasoning system, the Generalized Diagnosis Engine, G+DE, which is designed to compute solutions in the sense defined above, i.e. the minimal completions for maximal consistent assumption sets and the resulting quantity values. The proposed solution builds on existing algorithms for general constraint solving and GDE-style candidate generation. Our prototypical implementation employs a flexible commercial diagnostic engine, but it does not depend on any peculiarities of the program.

In the following, we present the key idea of the proposed approach, which consists of a transformation of the non-monotonic theory into sub-problems that can be computed monotonically. This approach is general in the sense that it relies only on minor restrictions on the class of domain theories that can be used as input (section 7.1). Further, the algorithmic design of all steps is discussed in detail (section 7.2) and the system architecture of the G+DE prototype is presented (section 7.3).

### 7.1 Key Idea of the Approach

From an abstract perspective, our approach is to compute the forward completion, as characterized above - and all occurring behavior constituents resulting from it - and then to use the GDE algorithm to simultaneously check its completeness and the consistency of the set of user-defined assumptions. In the case of inconsistencies, retraction of user-defined assumptions and/or backward completion of the structure are necessary to achieve valid solutions.

Unfortunately, obtaining a valid forward completion is not at all straightforward, due to the non-monotonicity inherent in the semantics: In collecting a set of structural elements as being part of the forward completion, one has to apply closure assumptions in order to compute values for the associated quantities (in the case of influence combinations) - but these values are in turn necessary to determine the occurrence of behavior constituents (via their quantity effects), which once again have an impact on the existence of other structural elements within the forward completion, thus challenging the closure assumption.

Similarly, checking of completeness, in the form of checking assumptions about model closure for their consistency, requires a smart algorithm, especially if we expect to obtain clues as to what kind of structural additions to search for in the case of an inconsistency.

Section 7.1.1 outlines the central idea of the algorithmic approach, the concept of the extended forward space. In section 7.1.2, a formalization of the problem transformation is presented and the main theorems for proving its equivalence to the original semantics are given. Section 7.1.3 gives an overview of the algorithmic design.

### 7.1.1 Computing Completions - The Extended Forward Space

Figure 7.1 shows how a structure initially specified in the situation description as provided by the user (shaded middle part) can be completed both by necessary effects (to the right) and by necessary additional causes (to the left of the situation description). The extensions of the predicates used in defining the
solution are indicated (KnownElement, EffectElement, and IntroducedElement, see section 6.2). Although
the diagram suggests this is mainly happening at a structural level (the boxes symbolize element
instances), note that the only way to discover the incompleteness of a situation description is when
influence resolution (based on the complete set of currently known influences, see section 5.3.3.6) yields
contradictory results. Thus, the quantity value assignments play an important role in completing the
situation description.

![Figure 7.1: Completion of situation description:
"forward" vs. "backward" and extensions of predicates](image)

The diagram is intended as an intuitive illustration. Causation mediated by behavior constituents is
indicated by rounded rectangles with the label "BC", arrows into them represent conditions and arrows
out of behavior constituents stand for (structural) effects.

Usually, the completion is not uniquely defined, especially when considering the interaction of multiple
backward completions with restrictions on parts of the original situation description as shown in figure
7.2.

![Figure 7.2: Multiple completions and the interaction between completions and restrictions](image)

The key idea is to compute a superset of all initial forward completions by neglecting part of the
occurrence conditions, namely the quantity conditions. Thus, we will collect all behavior constituents that
can potentially occur, as far as their structural conditions are concerned. At the same time we have to
collect their (potential) structural effects - and the behavior constituents that could result from these and
so on. As a main advantage, this superset, we will call it the *extended forward space*, can usually be constructed in a strictly monotonic fashion from the initial situation description specified by the user. See below (section 7.1.2 and appendix A) for a formal treatment of the necessary conditions for this construction. Figure 7.3 depicts the extended forward space:

![Figure 7.3: The "extended forward space" as a superset of the forward completion](image)

Since we may include behavior constituents (and their effects) that might never actually occur - because their quantity conditions are never met - we have to create a conditional structure to prevent these effects. By creating a specific set of constraints, we can ensure that the quantity effects of a given behavior constituent instance become effective if and only if its quantity conditions are met. In the same way, also the structural effects are "guarded" (for details refer to section 7.2.1). This separation of structural and quantity conditions corresponds to the distinction between "instantiation" and "activation" of processes in QPE. See section 7.2.1.1 for details about the difference to the distinction drawn by Forbus ([Forbus 1984]).

The resulting set of behavior constituents and the associated conditional structure is then checked for completeness, or more precisely: it is checked whether there is a solution contained in the extended forward space. This is achieved on the basis of standard constraint solving and assumption tracking. The hypothesis that there is a solution within the extended forward space is encoded as a set of "local" closed-world assumptions, which mark the precise points where the model closure (the "global" closed-world assumption) makes a difference: it is recorded whenever a value for a quantity is computed through a finite set of influences. Section 7.2.2 describes how influences are resolved into constraints labeled with the respective closed-world assumption. Optionally, the resulting constraint network can be transformed for purposes of simplification or with certain, mainly algebraic, enhancements (see section 7.2.5).

By employing a form of the classical GDE algorithm we can compute minimal sets of assumptions to be retracted - and closed-world assumptions provide helpful starting points for a "backward" search for structural additions - which are needed if the forward space does not suffice to find a solution (see section 7.2.4). In our implementation, this search is usually conducted as a computer-assisted interactive exploration of the potentially large search space.
7.1.2 Formalization of the Approach

For formalizing the chosen algorithmic approach, we employ an extended set of predicates. Instead of using the original definition of behavior constituents and the associated occurrence rules (rules (OR1) and (OR2), see section 5.3.3.3), we will rely on the definition of potential behavior constituents with instantiation and activation rules.

In section 7.1.2.1, we define basic predicates to be used in the following. Section 7.1.2.2 describes the construction of the extended forward space, using the initial situation description and instantiation rules. Finally, the activation rules for establishing actual occurrence and its effects are presented in section 7.1.2.3.

We provide evidence for the equivalence of this approach to the semantics stated in the previous section using a set of important theorems, which are proven in appendix A.

7.1.2.1 Basic Definitions - Potential Elements and Behavior Constituents

The extended forward space is constituted by structural elements that are potentially part of the forward completion, specified by

\[
\begin{align*}
\text{PotentialObject}(po) \\
\text{PotentialRelationTuple}(prt) \\
\text{PotentialElement}(e) \leftrightarrow \text{PotentialObject}(e) \vee \text{PotentialRelationTuple}(e) \\
\text{PotentialRelationTuple}_e(prt, rel, po_1, ..., po_n)
\end{align*}
\]

The last predicate is characterized in analogy to \(\text{RelationTuple}_e(rt, rel, o_1, ..., o_n)\) (see section 5.3.1.2). The definition of \(\text{IsOfType}\) is extended to \(\text{IsOfType}'\), which can be used for potential objects. Next, we extend the definition of mappings (cf. section 5.3.3.1) to cover potential objects, as well:

\[
\text{Maps}'(m, or, po) \rightarrow \text{Mapping}(m) \wedge \text{ObjectRole}(or) \wedge \text{PotentialObject}(po)
\]

with an analogous characterization w. r. t. uniqueness:

\[
\forall m_1 \forall m_2 \text{Mapping}(m_1) \wedge \text{Mapping}(m_2) \rightarrow
(\forall or \text{ObjectRole}(or) \forall po \text{PotentialObject}(po) (\text{Maps}'(m_1, or, po) \leftrightarrow \text{Maps}'(m_2, or, po))) \rightarrow
m_1 = m_2
\]

and the requirement that there exists at least a mapping for each configuration of potential objects matching the structural conditions:

\[
\text{StructuralConditionExObject}(bct, or_1, ot_1) \wedge ... \wedge \text{StructuralConditionExObject}(bct, or_n, ot_n) \wedge
\text{IsOfType}'(po_1, ot_1) \wedge ... \wedge \text{IsOfType}'(po_n, ot_n) \rightarrow
\exists m \text{Mapping}(m) \wedge \text{Maps}'(m, or_1, po_1) \wedge ... \wedge \text{Maps}'(m, or_n, po_n)
\]

The potential structural elements are introduced by (and give rise to) potential behavior constituents, characterized in analogy to \(\text{BehaviorConstituent}(bc, bct, m)\) (see section 5.3.3.3):

\[
\text{PotentialBehaviorConstituent}(pbc, bc, bct, m) \rightarrow \text{BehaviorConstituentType}(bc, bct) \wedge \text{Mapping}(m)
\]
\[
\text{PotentialBehaviorConstituent}(pbc_1, bc_1, m_1) \wedge \text{PotentialBehaviorConstituent}(pbc_2, bc_2, m_2) \rightarrow
(pbc_1 = pbc_2 \leftrightarrow bc_1 = bc_2 \wedge m_1 = m_2)
\]
Definition of the Extended Forward Space - Instantiation Rules

Now, the central instantiation rules for potential behavior constituents can be formulated:

**Instantiation rules:**

\[
\begin{align*}
\text{IR}_1 & \quad \text{PotentialSC}(bct, m) \rightarrow \exists pbc \exists m' \text{ Extends}(m', m, bct) \land \\
& \hspace{1cm} \text{PotentialBehaviorConstituent}(pbc, bct, m') \\
\text{PotentialBehaviorConstituent}(pbc, bct, m) & \rightarrow \text{PotentialSC}(bct, m) \\
\text{IR}_2 & \quad \text{PotentialBehaviorConstituent}(pbc, bct, m) \rightarrow \text{PotentialSE}(pbc)
\end{align*}
\]

with the helper predicates

\[
\begin{align*}
\text{PotentialSC}(bct, m) & \leftrightarrow \\
& \quad (\text{StructuralConditionExObject}(bct, or, ot) \rightarrow \\
& \hspace{1cm} \exists po \text{ PotentialObject}(po) \land \text{IsOfType}'(po, ot) \land \text{Maps}'(m, or, po)) \\
& \quad (\text{StructuralConditionExRelationTuplen}(bct, rel, or_1, ..., or_n) \rightarrow \\
& \hspace{1cm} \exists prt \text{ PotentialRelationTuple}(prt) \\
& \hspace{1cm} \exists po_1 \ldots \exists po_n \text{ PotentialObject}(po_1) \land \ldots \land \text{PotentialObject}(po_n) \land \\
& \hspace{1cm} \text{Maps}'(m, or_1, po_1) \land \ldots \land \text{Maps}'(m, or_n, po_n) \\
& \hspace{1cm} \text{PotentialRelationTuplen}(prt, rel, po_1, ..., po_n))
\end{align*}
\]

\[
\begin{align*}
\text{PotentialSE}(pbc) & \leftrightarrow \\
& \quad (\text{PotentialBehaviorConstituent}(pbc, bct, m) \rightarrow \\
& \quad (\text{StructuralEffectExObject}(bct, or, ot) \rightarrow \\
& \hspace{1cm} \exists po \text{ PotentialObject}(po) \land \text{IsOfType}'(po, ot) \land \text{Maps}'(m, or, po)) \\
& \hspace{1cm} (\text{StructuralEffectExRelationTuplen}(bct, rel, or_1, ..., or_n) \rightarrow \\
& \hspace{1cm} \exists prt \text{ PotentialRelationTuple}(prt) \\
& \hspace{1cm} \exists po_1 \ldots \exists po_n \text{ PotentialObject}(po_1) \land \ldots \land \text{PotentialObject}(po_n) \land \\
& \hspace{1cm} \text{Maps}'(m, or_1, po_1) \land \ldots \land \text{Maps}'(m, or_n, po_n) \\
& \hspace{1cm} \text{PotentialRelationTuplen}(prt, rel, po_1, ..., po_n)))
\end{align*}
\]

The first instantiation rule (IR₁) states that a potential behavior constituent is present if and only if the structural conditions are potentially fulfilled, i.e. fulfilled in terms of potential elements being present. According to the second instantiation rule (IR₂), the potential structural effects are created with the potential behavior constituent. Together, this amounts to the "instantiation" of a behavior constituent, see the next subsection (7.1.2.3) for the activation rules.

The starting point for the construction of the extended forward space is given by a "copy" of the initial situation description. Each structural element specified by the user is by definition equivalent to a potential structural element:

\[
\text{KnownElement}(e) \rightarrow \exists pe \text{ PotentialElement}(pe) \land \text{Equiv}(e, pe)
\]

The predicate Equiv maps from "actual" structural elements to potential ones in such a way that structural equivalence can be shown. This is defined in detail below.

The following important theorem provides the first part of the evidence, that the presented reasoning approach is equivalent to the original semantics:
Theorem 1:

If there is no backward completion, there exists a subset of the extended forward space that is structurally equivalent to the forward completion of the situation description. Formally:

$$(\forall e \neg \text{IntroducedElement}(e)) \rightarrow$$

$$(\text{BehaviorConstituent}(bc, bct, m) \rightarrow$$

$$\exists pbc \exists m' \text{PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc)) \land$$

$$(\text{Element}(e') \rightarrow \exists pe \text{PotentialElement}(pe) \land \text{Equiv}(e', pe))$$

The concept of structural equivalence between actual and potential elements, left implicit in the theorem, is defined as follows:

Structural Equivalence:

A relation $\text{Equiv}$, applicable to objects, relation tuples and behavior constituents - relating them to potential objects, potential relation tuples and potential behavior constituents, respectively - defines a structural equivalence, if it has the following properties:

(M) $\text{Equiv}(x, y_1) \land \text{Equiv}(x, y_2) \rightarrow y_1 = y_2$

(I) $\text{Equiv}(x_1, y) \rightarrow \text{Equiv}(x_2, y) \rightarrow x_1 = x_2$

(SE1) $\text{RelationTuple}(rt) \land \text{PotentialRelationTuple}(prt) \rightarrow$

$$(\text{Equiv}(rt, prt) \leftrightarrow \exists rel \exists o_1 \ldots \exists o_n \exists po_1 \ldots \exists po_n$$

$$\text{RelationTuple}_n(rt, rel, o_1, \ldots, o_n) \land$$

$$\text{PotentialRelationTuple}_n(prt, rel, po_1, \ldots, po_n) \land$$

$$\text{Equiv}(o_1, po_1) \land \ldots \land \text{Equiv}(o_n, po_n))$$

(SE2) $\text{BehaviorConstituent}(bc, bct_1, m_1) \land \text{PotentialBehaviorConstituent}(pbc, bct_2, m_2) \rightarrow$

$$(\text{Equiv}(bc, pbc) \leftrightarrow bct_1 = bct_2 \land$$

$$\forall or \text{Maps}(m_1, or, o) \land \text{Maps}'(m_2, or, po) \rightarrow \text{Equiv}(o, po))$$

Taken together, these properties require $\text{Equiv}$ to be an injective mapping (properties (I) and (M)) meeting certain requirements about structural equivalence, namely that relation tuples are equivalent if and only if all their parameters are (SE1), and that behavior constituents are equivalent if and only if all bound objects, both from the structural conditions and from the structural effects are equivalent (SE2). Theorem 1 states that such a mapping into the extended forward space exists and is complete in the sense of providing a counterpart for all structural elements and behavior constituents from the forward completion.

For the construction of the structural equivalence mapping, as part of the proof of theorem 1, please refer to appendix A. This is a more precise formulation of the idea expressed in figure 7.3 above, i.e. constructing a superset of the forward completion.

Note that the chosen algorithmic approach relies on the extended forward space to be uniquely defined - i.e. there is no ambiguity in the unification of created objects. When modeling carefully, this can be achieved for many application domains by employing unique and functional relations. See section 5.3.1.2 for a discussion of these modeling techniques and note that all of the application examples in section 8 naturally meet this requirement. Technically, the construction of the extended forward space is then possible in a monotonic fashion. The logical definition of this property in the form of unification rules can be found in appendix A.
7.1.2.3 Reasoning Within the Extended Forward Space - Activation Rules

A potential behavior constituent requires two additional conditions to be fulfilled to actually occur, i.e. to become active: Obviously, the quantity conditions have been neglected, but also the structural conditions so far merely reference potential structural elements - it has to be checked, whether they are actually fulfilled. We employ a guard predicate, exists, for all structural elements, to distinguish between their potential and their actual existence. The same predicate is also used for potential behavior constituents to indicate that their structural conditions are actually fulfilled. A second guard predicate, active, determines whether the potential behavior constituent actually occurs - and imposes its defined effects. See the following rules for the interdependencies:

Activation rules:

\[(AR_1)\] PotentialBehaviorConstituent(pbc, bct, m) → (
\((\forall pe \text{ IsSCElem}'(pe, pbc) \rightarrow \text{exists}(pe)) \leftrightarrow \text{exists}(pbc))\)

\[(AR_2)\] PotentialBehaviorConstituent(pbc, bct, m) → 
(QC'(bct, m) ∧ exists(pbc) ↔ active(pbc))

\[(AR_3)\] PotentialBehaviorConstituent(pbc, bct, m) → 
(active(pbc) → (\(\forall pe \text{ IsSEElem}'(pe, pbc) \rightarrow \text{exists}(pe)\))

\[(AR_4)\] PotentialBehaviorConstituent(pbc, bct, m) → 
(active(pbc) → QE'(pbc))

with the following helper predicates

\[\text{IsSCElem}'(pe, pbc) \leftrightarrow \text{IsSCObject}'(pe, pbc) \lor \text{IsSCRelationTuple}'(pe, pbc)\]

\[\text{IsSCObject}'(po, pbc) \leftrightarrow \exists or (\exists ot StructuralConditionExObject(bct, or, ot) ∧ Maps'(m, or, po))\]

\[\text{IsSCRelationTuple}'(prt, pbc) \leftrightarrow \exists rel (\exists or1 ... \exists orn StructuralConditionExRelationTuple_n(bct, rel, or1, ..., or_n) ∧ Maps'(m, or1, po1) ∧ ... ∧ Maps'(m, orn, pon))\]

\[\text{IsSEElem}'(pe, pbc) \leftrightarrow \text{IsSEObject}'(pe, pbc) \lor \text{IsSERelationTuple}'(pe, pbc)\]

\[\text{IsSEObject}'(po, pbc) \leftrightarrow \exists or (\exists ot StructuralEffectExObject(bct, or, ot) ∧ Maps'(m, or, po))\]

\[\text{IsSERelationTuple}'(prt, pbc) \leftrightarrow \exists rel (\exists or1 ... \exists orn StructuralEffectExRelationTuple_n(bct, rel, or1, ..., or_n) ∧ Maps'(m, or1, po1) ∧ ... ∧ Maps'(m, orn, pon))\]

The predicates QC and QE have been extended (to QC’ and QE’) to accommodate quantities associated with potential objects. The first activation rule (AR_1) determines the fulfilling of the structural conditions from the "existence" of the individual elements. The second activation rule (AR_2) states that a potential behavior constituent is active if and only if its structural conditions are actually fulfilled (exists(pbc)) and so are its quantity conditions. (AR_3) establishes the structural effects of active behavior constituents, in the form of implying the actual existence of the individual elements again. Finally, (AR_4) does the same
for the quantity effects. Section 7.2.1 gives a detailed account of how the respective conditions and effects are encoded.

Again, there is a starting point for determining existence and activity: The only potential structural elements in the extended forward space known to exist are the ones from the initial situation description. More precisely, we collect the potential elements that are structurally equivalent to the situation description:

\[
\text{KnownElement}(e) \land \text{Equiv}(e, pe) \implies \exists(\text{pe})
\]

Now, we present the second important theorem for proving the correctness of the computational approach. Theorem 2 is a confirmation of the design goal that the all actual behavior constituents are identified and activated and all actual structural elements are determined to exist:

**Theorem 2:**

If a potential behavior constituent or a potential structural element is part of the structurally equivalent subset defined in theorem 1 (i.e. it has an actual counterpart), it is active or "existing", respectively. Formally:

\[
\begin{align*}
\text{PotentialBehaviorConstituent}(pbc, bct, m) & \implies \\
(\exists bc \exists m' \text{BehaviorConstituent}(bc, bct, m') \land \text{Equiv}(bc, pbc)) & \implies \text{active}(pbc) \\
\text{PotentialElement}(pe) & \implies \\
(\exists e \text{Element}(e) \land \text{Equiv}(e, pe)) & \implies \exists(\text{pe})
\end{align*}
\]

The proof is to be found in appendix A.

This proves that all required elements are activated. However, the activation rules in the given form do not exclude the existence of elements without a supporting (active) behavior constituent - formally, these would be a part of the backward completion. It makes sense to exclude this for the extended forward space by using a stricter version of the third activation rule: We add the requirement that for an element to exist, it has to be either part of the initial situation description or the structural effect of an active behavior constituent. Formally, the following rule will be added:

\[
\begin{align*}
(\text{AR}_{3b}) & \quad \text{PotentialElement}(pe) \implies \\
(\exists e \text{KnownElement}(e) \land \text{Equiv}(e, pe)) & \lor \\
(\exists pbc \exists bct \exists m \text{PotentialBehaviorConstituent}(pbc, bct, m) & \land \\
\text{active}(pbc) \land \text{IsSEElem}'(pe, pbc))
\end{align*}
\]

This requires every structural element of the extended forward space to be a "known" or an "effect" element. For the sake of simplicity, we can write a single improved third activation rule, which is a consequence of (AR₃) and (AR₃b):

\[
\begin{align*}
(\text{AR}_{3'}) & \quad \text{PotentialElement}(pe) \implies \\
(\exists e \text{KnownElement}(e) \land \text{Equiv}(e, pe)) & \lor \\
(\exists pbc \exists bct \exists m \text{PotentialBehaviorConstituent}(pbc, bct, m) & \land \\
\text{active}(pbc) \land \text{IsSEElem}'(pe, pbc))
\end{align*}
\]

Using this stricter rule and for the case that there no loops in the creation structure, i.e. the structural effects of behavior constituents do not lead to the creation of their own structural conditions (this excludes the pathological cases mentioned in section 6.2.1), we can prove a stronger theorem. Note that
only loops in the creation structure are excluded, quantity effects with arbitrary ramifications are allowed, of course. For most applications, this can be generally established from the form of the domain theory.

**Theorem 2b:**

If there are no loops in the creation structure, then potential behavior constituents are active and potential structural elements exist, if and only if they are part of the structurally equivalent subset defined in theorem 1 (i. e. they have actual counterparts). Formally:

\[
\text{PotentialBehaviorConstituent}(pbc, bct, m) \rightarrow \\
(\exists bc \exists m' \text{ BehaviorConstituent}(bc, bct, m') \land \text{Equiv}(bc, pbc) \leftrightarrow \text{active}(pbc))
\]

\[
\text{PotentialElement}(pe) \rightarrow \\
(\exists e \text{ Element}(e) \land \text{Equiv}(e, pe) \leftrightarrow \text{exists}(pe))
\]

Once more, we provide a formal proof in appendix A. As a remark, the additional rule (AR\textsubscript{3b}) is very similar to the influence closure axiom (IC) from section 5.3.5 - and we employ a very similar mechanism for effectively enforcing it in the composed behavior model and for detecting potential violations, i. e. a special kind of influence resolution with a special CWA, see sections 7.2.1.4 and 7.2.2.2.

For a full proof of equivalence between the semantics defined in sections 5 and 6 on the one hand and the transformation into a monotonically constructible extended forward space followed by constraint propagation (and potentially search for structural revisions) on the other hand, a detailed analysis of the encoding of quantitative effects would be required. In particular, it is necessary to ensure that inactive behavior constituents do not have any discernible effect on object quantities and structural elements. We omit the (lengthy and rather mechanical) proof of this equivalence, and continue with the presentation of the details of the algorithmic design, which should provide sufficient evidence.

### 7.2 Algorithmic Design

The algorithmic design of \( \text{G}^*\text{DE} \) is a realization of the generalized theory of consistency-based diagnosis presented in section 4. We repeat the diagram giving an overview of consistency-based problem solving in figure 7.4.

Remember that even the specialized GDE algorithm is an instance of this general principle - only with very limited capabilities in model composition and model revision. We decided to rely on known and well-developed algorithms for constraint propagation, dependency recording and conflict generation, as found in an appropriate GDE implementation.

On top of this basic GDE algorithm, we create the infrastructure for model composition and model revision as needed for the much more general process-oriented modeling paradigm and the kind of solutions to be computed. For model composition, we rely on the presented transformation of the non-monotonic problem into the construction of the extended forward space, i. e. collecting all potential behavior constituent instances with the addition of special model closure assumptions. Model revision includes backward search for additions to the structure that can explain contradictions to these closure assumptions.
In the following, the individual steps are described in detail. Section 7.2.1 treats model composition in terms of determining the extension of the extended forward space with particular focus on how the conditions and effects of behavior constituents are encoded, so that they are activated if and only if the activity conditions are met. The resulting Extended Qualitative Influence Diagram (XQID) needs to be resolved and, thus, enhanced with closed-world assumptions and transformed into a constraint network (see section 7.2.2), so that it can be submitted to a consistency check by standard GDE procedures (section 7.2.3). It is the inclusion of "local" closed-world assumptions that turns the consistency check into a completeness check at the same time - and provides a good starting point for model revision, in case these assumptions show up in conflicts. Section 7.2.4 is about the structure completion algorithm, which carries out the "backward" search for missing behavior constituent instances and the structural elements from their respective structural conditions. It is a generalization of the module generating the extended forward space in the first place. Finally, optional model transformations and especially model enhancements are discussed in section 7.2.5, which turn out to be extremely valuable for practical applications. Most of the application examples in section 8 rely on some form of model enhancement.

7.2.1 Model Composition

As an important difference to the component-oriented case, we find that in the generalized approach "model composition" becomes an issue. Many benefits of a compositional modeling approach, such as reusability and easy adaptation for new device configurations, can already be seen in the specialized component-oriented GDE problem domain. But the behavior model used for diagnosis is basically fixed for the device: one can collect the (default) component models, connect them according to shared terminals and use the resulting constraint network for diagnosis. Also the search for consistent models is organized as relatively simple mode switches in GDE⁺ ([Struss/Dressler 1989]), Sherlock ([de Kleer/Williams 1989]), or DDE ([Dressler/Struss 1994]), i.e. most of the system model remains fixed.

In process-oriented modeling and, hence, in the generalized case of G*DE models, however, the set of behavior constituents occurring in a given situation requires reasoning on its own. On top of that, testing a hypothesis that introduces completely new elements into the system structure, clearly necessitates the ability to compose a behavior model for the extended system automatically, i.e. without human intervention.
7.2.1.1 Construction of the Extended Forward Space

As mentioned before, the idea of the extended forward space corresponds to a distinction between instantiation and activation of behavior constituents, as proposed by Forbus ([Forbus 1984]). Forbus, however, treats the distinction as an ontological one, or at least has embedded it in the semantics of QPT. We take the perspective that an instantiated but inactive process is at most an algorithmic construct for calculating valid solutions of a given problem. This is supported by the observation that the distinction between instantiation and activation conditions is mostly arbitrary. Actually, our implementation and the original description of QPT make a different cut here with respect to some structural conditions: In QPT, certain predicates and relations can be declared as "preconditions" - so they are considered for activation, but disregarded during instantiation ([Forbus 1984, p. 98]).

We propose that at least the complete set of structural conditions should be used as instantiation conditions in order to avoid the intricate problems arising from the non-monotonic feedback loop described above, but the algorithm is open for pre-calculating some quantity conditions (especially for constant values), so that some certainly inactive behavior constituents need not be included in the model at all. When collecting all potentially active behavior constituents, each configuration of elements from the structure, i.e. the scenario extended by already (potentially) created elements, matching the structural conditions of a behavior constituent type gives rise to an instance of this particular type. However, in handling the structural effects, we have to observe important unification issues. Namely, the relations a particular object from the structural effects is participating in, might be of a unique or function type, thus determining that there can be only a single object instance filling a particular place. From that we might infer that a structural effect is already fulfilled, i.e. the object role unifies with an existing instance rather than creating a new one (compare the discussion of the semantics of structural effects in section 5.3.3.4).

In the current design of the Generalized Diagnosis Engine, we have decided to restrict ambiguous unifications. In fact, we will generate a single extended forward space, rather than multiple ones, which would require to combine the different results in a sophisticated way afterwards. It turns out that most well-designed domain theories allow to determine whether an object has to be created or is to be identified as an existing one. The user will be warned if a situation produces an ambiguous unification under the given domain theory. The proofs in appendix A rely on this restriction, which is also formally defined there.

The structural completion algorithm, which forms the basis of the composition module, will be discussed in detail below (section 7.2.4.2). It is designed to incrementally collect all potential behavior constituent instances created from fulfilled structural conditions (forward completion) as well as to search potential behavior constituent instances achieving specified structural or quantity effects (backward completion). The next section will describe how the conditional structure for a created behavior constituent instance is built.

This will be done by creating an abstract data structure called Extended Qualitative Influence Diagram, or XQID. The original form, Qualitative Influence Diagram, or QID, basically consists of variables and typed influences between them. It is extended here in that also constraints can connect variables. For a precise definition of QIDs, please refer to [Heller/Struss 1996], where they have been used as the basis for syntactical model transformations in various respects. Section 7.2.5 will discuss shortly, how one can profitably apply model transformations to different intermediate results of the overall reasoning process.
7.2.1.2 Encoding Structural Conditions

Now, we will take a closer look at how to "guard" an instantiated potential behavior constituent, so that its effects have an impact if and only if the behavior constituent actually occurs. We will use the example from section 5.3.3 to illustrate the construction of the conditional structure. The diagram in figure 7.5 shows an instance, IronRedissolving1, of the behavior constituent type IronRedissolving, i.e. the bound objects rather than the roles are shown and all quantity names are given in fully qualified version:

![Diagram](attachment:image.png)

**Figure 7.5: An instance of the IronRedissolving behavior constituent from section 5.3.3.**

The semantics require that both the quantity and the structural conditions are actually fulfilled. Note that this has to be confirmed for the potential structural elements from the structural conditions, as well. We use boolean guard variables corresponding to the guard predicate exists, introduced in section 7.1.2.3, for example:

- IronSediment.exists
- dissolved-in[IronWater,Layer].exists
- IronRedissolving1.exists

For the sake of simplicity, we create these variables for each structural element, regardless of whether it has been specified by the user or (potentially) created by a behavior constituent. This will simplify the creation of the conditional structure for a behavior constituent instance. Analogously, the specification of exists(pe) for a potential element, pe, that is equivalent to a part of the initial situation description (i.e. \( \exists e \text{ Equiv}(e, pe) \land \text{KnownElement}(e) \)) takes the form of a value assignment

\[ \text{pe.exists} = \text{true} \]

which is a convenient statement to attach a user-defined assumption to, if there is one. See the discussion of ATMS control in our prototype (section 7.3). Remember that the exists predicate for a potential behavior constituent instance, pbc, is used to encode the fulfilling of its structural conditions (compare first activation rule (AR1)).
Thus, a conjunction of the existence of all potential structural elements, \( pe_1 \) through \( pe_n \), occurring in the structural conditions of the behavior constituent is encoded, defining the "existence" of the behavior constituent itself:

\[
\text{exists}(pe_1) \land ... \land \text{exists}(pe_n) \leftrightarrow \text{exists}(pbc)
\]

becomes the constraint

\[
\text{AND}_n(pbc \text{.exists, } pe_1 \text{.exists, } ..., \ pe_n \text{.exists})
\]

The diagram in figure 7.6 illustrates the guard variables and the constraint created in the example:

![Figure 7.6: Guard variables and constraints for the structural conditions](image)

### 7.2.1.3 Encoding Quantity Conditions

For encoding quantity conditions, the set of defined constraint types is important: For each constraint type, \( \text{cstrType} \), occurring as a quantity condition, we expect another constraint type, \( \text{CONDITION-cstrType} \), to be defined, that uses the original parameters plus a boolean "result". This result, \( \text{res} \), is restricted to \( \text{true} \) if the original constraint holds for the remaining parameter set and \( \text{false} \) otherwise (see section 7.2.1.5 for an example). Logically written:

\[
\text{ConstraintSatisfied}((\text{CONDITION-cstrType}, \text{res}, v_1, ..., v_m)) \leftrightarrow (\text{res} = \text{true} \leftrightarrow \text{ConstraintSatisfied}(\text{cstrType}, v_1, ..., v_m))
\]

For each quantity condition that applies the constraint type \( \text{cstrType} \) to quantities \( qu_1 \) through \( qu_m \), we add a variable \( q_c \) and the constraint

\[
\text{Constraint}((\text{CONDITION-cstrType}, q_c, qu_1, ..., qu_m))
\]
Now we can express the activity of a potential behavior constituent, pbc.active, using the conjunction of all quantity condition results, qc1 through qck, and the "existence" of the potential behavior constituent, pbc.exists:

\[ \text{AND}(\text{pbc.active}, \text{pbc.exists}, \text{qc}_1, \ldots, \text{qc}_k) \]

This constraint is the encoding of the second activation rule ((AR2) see section 7.1.2.3). Note that the equivalence in the activation rule is the key to reasoning about conditions that have to be met for effects to occur. Figure 7.7 shows the encoding of quantity conditions in the IronRedissolving example:

![Figure 7.7: Guard variables and constraints for encoding the quantity conditions](image)

### 7.2.1.4 Encoding Structural Effects

The activity variable, pbc.active, plays the role of the central switch for all effects of the potential behavior constituent. If it has the value true, all elements from the structural effects are to be created and all constraints and influences specified as quantity effects will be enacted (see activation rules (AR3) and (AR4) defined in section 7.1.2.3). However, if the potential behavior constituent is inactive (pbc.active = false), it has to be ensured that there is no impact on structure or quantity values whatsoever.

For structural effects, the existence variable will have to be implied by the activity variable. This does not exclude other ways of establishing the respective element, since multiple behavior constituents might force the very same thing into existence (see the discussion of unification of structural effect elements in section 7.2.1.1).

Nevertheless, we would like to be able to conclude that an existing element that was not specified by the user has to be supported by at least one occurring (i.e. active) behavior constituent. Thus, we ultimately want a stronger proposition than a simple set of implications for an element's existence variable, elem.exists from the activities of all supporting potential behavior constituents, pbc1 through pbcn, as in

\[ \text{IMPLIES}(\text{pbc}_1, \text{active}, \text{elem.exists}) \]

\[ \ldots \]

\[ \text{IMPLIES}(\text{pbc}_n, \text{active}, \text{elem.exists}) \]
Instead, the existence of the element - if not among the set of "known" elements from the initial situation description - also implies one of its creating behavior constituents to be active:

\[ \text{OR-n(elem.exists, pbc}_1\text{.active, ..., pbc}_n\text{.active)} \]

This is an encoding of the stronger activation rule (AR') (see section 7.1.2.3). Note that the above constraint cannot be created before the set of all supporting behavior constituents is known. Hence, the part derived from (ARb) relies on the model closure:

\[ \text{elem.exists} \rightarrow \text{pbc}_1\text{.active} \lor ... \lor \text{pbc}_n\text{.active} \]

We want to associate a closed-world assumption with the corresponding constraint, expressing that additional elements could invalidate this implication. This is very similar to the semantics of influences as defined in section 5.3.3.5 - and in fact, we will employ the same technique: In the instantiation phase, an influence of a special type ("implication influence") will be created and will be resolved later (see influence resolution in section 7.2.2). The implication symbol in a box with an arrow head in figure 7.8 denotes this kind of influence:

![Figure 7.8: Guard variables and "implication influences" for encoding the structural effects](image)

### 7.2.1.5 Encoding Quantity Effects

Finally, we have to guard the quantity effects. In the case of constraints, it is assumed that for each constraint type, \( \text{cstrType} \), there exists a derived type with an additional parameter. This time, the derived constraint, \( \text{CONDITIONAL-cstrType} \), is designed to impose the same restrictions on the original parameters, \( q_1 \) through \( q_n \), as the original constraint, if the additional "condition" parameter, \( \text{cond} \), is true. It is to allow any combination of parameter values, if \( \text{cond} = \text{false} \). Formally:

\[
\text{ConstraintSatisfied(CONDITIONAL-cstrType, cond, v_1, ..., v_n)} \leftrightarrow \\
((\text{cond} = \text{true}) \rightarrow \text{ConstraintSatisfied(cstrType, v_1, ..., v_n)})
\]
The difference to the derived type \texttt{CONDITION-cstrType} (see section 7.2.1.3 above) might seem minimal, but is an important one. We explain the distinction using the example of the simple constraint type \texttt{ZERO}, as defined by

\[
\text{ConstraintSatisfied}(\text{ZERO}, v) \iff (v = \text{zero})
\]

For \(v \in \{\text{neg, zero, pos}\}\) and a boolean variable, \(b \in \{\text{true, false}\}\), we can write the following normal forms for the derived constraint types:

\[
\begin{align*}
\text{ConstraintSatisfied}(\text{CONDITIONAL-ZERO}, b, v) & \iff \\
(b = \text{true} \land v = \text{zero}) & \lor (b = \text{false} \land v = \text{neg}) \lor \\
(b = \text{false} \land v = \text{pos}) & \lor (b = \text{false} \land v = \text{zero})
\end{align*}
\]

\[
\begin{align*}
\text{ConstraintSatisfied}(\text{CONDITION-ZERO}, b, v) & \iff \\
(b = \text{true} \land v = \text{zero}) & \lor (b = \text{false} \land v = \text{neg}) \lor \\
(b = \text{false} \land v = \text{pos})
\end{align*}
\]

The difference lies in the last conjunction of the first expression: The \textit{conditional} constraint type imposes no restriction on the original parameters (here only \(v\)), when the condition, \(b\), is false, so the original constraint might as well be fulfilled (\(v = \text{zero}\)). The \textit{condition} version of a constraint type, however, restricts the result (also \(b\) here) to true, whenever the original constraint is fulfilled.

Using the defined new constraint type, the desired semantics can be encoded by using the activity variable, \texttt{pbc.active}, as a condition on each quantity effect constraint (see figure 7.9 below).

In the case of influences, we expect each type of influence, \texttt{inflType}, to possess a value, \texttt{neutral\_inflType}, that acts as a "neutral contribution", i.e. its inclusion in the summary resolution constraint (see next section) will not change the result. For algebraic combinations, this corresponds to the notion of a neutral element, such as for additive influence combination, a "zero" element will not change the sum.

Now, we define a special ternary constraint, \texttt{COND-EQUAL-inflType}, as follows:

\[
\begin{align*}
\text{ConstraintSatisfied}(\text{COND-EQUAL-inflType}, \text{cond}, v_1, v_2) & \iff \\
((\text{cond} = \text{true}) \land (v_2 = v_1)) & \lor ((\text{cond} = \text{false}) \land (v_2 = \text{neutral\_inflType}))
\end{align*}
\]

By creating a new guard variable, \texttt{pbc.guardi}, for each influence from the quantity effects, we can ensure that the respective influence makes either the original contribution or a neutral one, depending on the activity of the potential behavior constituent. Thus, an influence

\[
\text{Influence(inflType, qu}_1, \text{qu}_2)
\]

becomes

\[
\begin{align*}
\text{Constraint(\text{COND-EQUAL-inflType, pbc.active, pbc.guardi, qu}_1)} & \\
\text{Influence(inflType, pbc.guardi, qu}_2)
\end{align*}
\]

In figure 7.9, the resulting variables and constraints are shown for the example (\texttt{IronRedissolving1.active} is fed to the three new constraints as the condition parameter):
The complete influence diagram fragment as generated from the behavior constituent instance can be seen in figure 7.10. The objects and relation tuples are still shown as dashed boxes and ellipses, but they are not part of the constraint network and only represent a skeleton for the construction.

We emphasize once again the openness of the modeling language to the employment of almost arbitrary constraints and even influence types. All that is required for the model composition phase is a basic set of constraint types and for each constraint type used in quantity conditions or quantity effects a derived one that yields a result or accepts a condition, respectively. The constraint types employed internally are conjunction and disjunction with an arbitrary number of parameters - but see section 7.2.5.1 for how high-
arity constraints can be broken down later), and boolean equivalence, as well as for influence resolution
the respective summary constraints (see below) and a "conditional equality" yielding a neutral
contribution to such a sum. This provides significant freedom for the modeler in creating constraints and
even new kinds of influences - provided the constraint solver used with the GDE core can handle them.

7.2.2 Influence Resolution and the Closed-World Assumption

In the first step, existing quantities and new guard variables were connected by constraints and influences.
During the model composition phase, the actual objects and relation tuples comprising the structural
skeleton for the quantities lost their significance for the behavior model. The resulting data structure has
been defined as an XQID. As a next step, all influences contained in the XQID will be resolved into finite
resolution constraints, which, by definition, is possible for the complete model (see section 5.3.3.6 for the
semantics).

The core problem is the determination of the completeness of the model. Our approach relies on
assumption-based reasoning, and by making the closure of the model an explicit assumption, it can be
retracted as part of the diagnostic process. We can precisely define the global closed-world assumption
(CWA) used:

\[
\text{CWA} \iff (\forall e \neg \text{IntroducedElement}(e))
\]

This states that there are no "introduced" elements, i.e. the backward completion is empty, or
equivalently (see theorem 1), there is a solution within the extended forward space. This is the central
hypothesis we start with and that can lead us to minimal consistent structures. But from contradictions to
the global CWA and its retraction, we are also able to conclude the existence of introduced elements,
which then requires "backward" search for these elements (see below).

In the following, we examine the consequences of the global CWA for influence resolutions (section
7.2.2.1) and for the implication of structural effects (section 7.2.2.2), leading to the creation and usage of
"local closed-world assumptions".

7.2.2.1 Local Closed-World Assumptions for Quantities

First we will see the importance of the closed-world assumption for each individual resolution of
influences on a common target quantity. For this, we require that all influences are known. When taking
the set of all influences in the extended forward space, we can define what we call the local closed-world
assumption for a given quantity, \( q_u \), using the set of influences from quantities \( q_{u1} \) through \( q_{un} \) directed
to it:

\[
\text{CWA}_{q_u} \iff (\text{Influence}(it, qu', qu) \rightarrow (qu' = qu_{i}) \lor ... \lor (qu' = qu_{n}))
\]

Examining the relation of this assumption to the global CWA, we look at theorem 1 (section 7.1.2.2),
which states that the validity of the global CWA in the sense defined above implies that the extended
forward space contains (the structural equivalent of) all elements actually present and all behavior
constituents occurring. With the closure of the domain theory (see section 5.3.5), we have presented the
important closure axiom for influences (IC), stating that all influences result from the quantity effects of
behavior constituents. Together with the CWA, this restricts the set of all influences to be contained in the
extended forward space. Hence, we have evidence for

Theorem 3:

\[
\text{CWA} \rightarrow (\forall qu \text{ Quantity}(qu) \rightarrow \text{CWA}_{q_u})
\]
The global closed-world assumption implies each local one (which is defined only semi-formally w. r. t. to the current extension of the extended forward space here). Therefore, we could associate the global CWA with each individual resolution constraint, so that its use is recorded for each prediction relying on such a summary constraint. In this manner, we would determine which value assignments depend on the closed-world assumption. However, we expect this to be the case for almost any prediction from given values - a fact often neglected in simplified simulation approaches creating the impression it was obvious, where the "system boundary" is to be drawn.

In fact, the appearance of the global CWA in a conflict (see section 7.2.3) would tell us *that* we have to open the model, but very little about *how* to do this and what kind of structural augmentation to look for. A more sophisticated approach is to distinguish each application of the closed-world assumption as to its consequence for the given target variable, as this is where the CWA "makes a difference".

Therefore, we associate the respective local closed-world assumption $\text{CWA}_{qu}$ with a resolution constraint. Note that by simple contraposition of theorem 3, we can conclude the existence of additional structural elements from the refutation of a single local closed-world assumption. But even more so, we have important additional information as to the effects of the structural element we are looking for. Namely, it has to fulfill the structural conditions of a behavior constituent instance that creates an influence on the given quantity, $qu$. The specifications of the quantity (quantity role and associated object type) provide valuable criteria for restricting the search for such behavior constituents. Section 7.2.4 presents a search algorithm that takes a refuted local CWA as an input and identifies all possible structural augmentations that could be responsible for breaking the closure.

An issue we have neglected so far is that the global closure assumption restricts the set of influences that are *actually in effect* - in contrast to those influences belonging to inactive behavior constituents. Thus, in the finite disjunction of the definition of $\text{CWA}_{qu}$ above, we could have superfluous elements, which is important only for determining when the local CWA is actually violated. But we have taken provisions by supplying neutral contributions for all inactive behavior constituents (see section 7.2.1.5) and this leads to the same results as if we had collected only the "active influences". Hence, we can use the closure of the extended forward space and consequently collect all potentially active influences into a summary constraint.

As a side remark, Collins ([Collins 1993], see section 9.1.4 for a discussion) uses local closed-world assumptions in a very similar way in his PDE system, although his definition of model closure is intricately connected to the justification structure managed by the underlying ATMS.

Another issue to be discussed here concerns influence resolution for uninfluenced quantities. It is clear that an influenced quantity will have to obey the resolution constraint (and we have pointed out that the modeler has to take care that all constraints directly introduced as quantity effects are also valid under these circumstances). However, resolution for quantities that are not influenced amounts to assigning a default value to the derivative, usually zero. This is not always desirable, as many unspecified "exogenous" quantities definitely should assume other reasonable values as part of state-completion (prediction), without violating a closed-world assumption. In other cases, the absence of influences should indeed restrict the value and lead to a contradiction to other observations.

There are different ways to handle this distinction. One is to require an explicit specification of "open" versus "closed" quantities by the user. Schemes for labeling certain quantity roles of certain object types are conceivable. This might seem awkward for the inexperienced user, but certainly provides the maximum flexibility for the experienced modeler.
Another possibility is to assume that an uninfluenced variable is always exogenous, i.e. "open". By using dummy influences, one can always close a given class of quantities. This is not as stringent as an explicit labeling, but allows for almost the same degree of flexibility. In fact, this is the scheme currently implemented in our prototype with the additional useful heuristic to close all derivative quantities introduced by way of automated model enhancement (see section 7.2.5.3), which results in the default \( \frac{d}{dt} q_u = \text{const.} \). In effect, an exogenous quantity can assume any value, but cannot change without an influence from an active behavior constituent. Note the similarity to the "sole mechanism assumption" in [Forbus 1984].

7.2.2.2 Local Closed-World Assumptions for Structural Elements

For structural effects (see section 7.2.1.4), the encoding of the additional activation rule \( \text{(AR}_{3b} \text{)} \) (see section 7.1.2.3) relies on the model closure, as well. Namely, the creation of a structural element (encoded as assigning a value of true to the associated exists variable) requires an active potential behavior constituent featuring the element in its structural effects.

Again, the set of potential behavior constituents collected into the extended forward space has to be determined as complete. The global CWA as defined above has been shown to be the key to this closure. Again, we can define a local closed-world assumption, this time for a potential structural element, \( \text{pe} \), stating that there is no additional behavior constituent supporting it, besides the known set, \( \text{pbc}_1 \) through \( \text{pbc}_n \):

\[
\text{CWA}_\text{pe} \iff (\text{PotentialBehaviorConstituent(pbc, bct, m)} \land \text{IsSEElem}'(\text{pe}, \text{pbc})) \rightarrow (\text{pbc} = \text{pbc}_1) \lor \ldots \lor (\text{pbc} = \text{pbc}_n))
\]

With an argument analogous to the one presented above, we can see that each instance of such an assumption follows from the global CWA:

**Theorem 3b:**

\[
\text{CWA} \rightarrow (\forall \text{pe PotentialElement(pe) } \rightarrow \text{CWA}_\text{pe})
\]

Again, contraposition tells us that an "existing" potential element without a supporting behavior constituent requires structural additions. This time, however, the extra information that can be gained from refuting a local \( \text{CWA}_\text{pe} \), consists of a particular element identified as unexpected. There is two possible solutions: the potential element, \( \text{pe} \), is itself part of the backward completion, i.e.

\[
\exists e \text{Equiv}(e, \text{pe}) \land \text{IntroducedElement(e)}
\]

or there is another supporting behavior constituent, not yet included in the extended forward space. The latter possibility triggers backward search again, see section 7.2.4.

To enable this, the resolution algorithm creates the special resolution constraint for the "backward implication" for structural effects (cf. section 7.2.1.4) and associates the local \( \text{CWA}_\text{pe} \) with it. It is the task of the consistency check to collect evidence for potentially refuting it.

7.2.3 Consistency Check

Influence resolution produces a pure constraint network with assumptions associated with some constraint instances. This is a typical input for a GDE-style diagnostic system, which can handle the standard tasks of prediction (constraint propagation) with assumption tracking, conflict detection and candidate generation. In GDE terminology, the constraint network is the "system description" (cf. section 3.1).
As a second input, quantity value assignments are supplied - these are labeled "observations" from the GDE perspective. In our case, quantity value assignments can be associated with assumptions, too - in fact, it is the place, where all user-defined assumptions go (see section 7.3). Assumed structural elements are marked by assigning a value of true to their existence variable - under the respective assumptions.

For this description, we assume there is a sufficiently efficient GDE implementation available (this is a serious requirement, since we might create large constraint networks), meeting the following requirements:

- It will accept some form of an externally generated constraint network with assumptions. It is acceptable to package it into a "component model" if that is the required form - but note that assumptions will have to be associated with individual constraints.
- Observations can be supplied with assumptions as well (this might require external manipulation of an ATMS used by the diagnostic engine).
- The integrated constraint solver will be able to handle all constraint types used in model composition. Ideally, the employed implementation allows for the explicit definition of new constraint types (this can be in the form of tables for small finite constraints or combinations of simpler constraint types).
- The quality of the generated results will depend on the completeness of the constraint solver. Typically, efficient implementations of diagnostic algorithms suffer from minor restrictions w. r. t. constraint solving, e. g. in the propagation of disjunctions. The worst effect of incomplete constraint solvers would be missing conflicts that could lead the system to incorrectly labeling an extension as consistent. But see the application in section 8.3.2 for an example of how to overcome this limitation by recognizing and exploiting structural configurations.
- Candidate generation typically relies on some cardinality focus, i. e. the smallest candidates are generated first. This is okay for finding minimal backward extensions, but note that the output of GDE is just an intermediate result and the set of all ultimately discovered solutions (one might call them "GDE candidates") might have to be ranked by different criteria (see section 6.3).

For illustrative examples of how constraint propagation with assumption tracking can achieve the detection of conflicts see the first two application examples in section 8, where we have included diagrams of this procedure (figures 8.13 and 8.24).

Another issue to be discussed here is the strictly state-based perspective taken in the diagnostic approach so far. The strict interpretation of the behavior constituent laws is as describing the "completion" of a state of the system under consideration ("if X is the case, then also Y is (already) the case"). Reasoning about action and change (in the form of "if now X is the case, at some point in the future Y will be the case") requires more sophisticated techniques. However, the diagnostic potential derived from detecting inconsistencies within a single state is not to be underestimated: under certain conditions, it can be proven that this approach is equivalent to a simulation-based one ([Struss 1997]).

It is ongoing research whether the usage of a more powerful (temporal) diagnostic engine at the core (e. g. based on MCTCP [Dressler/Freitag 1994]) is compatible with the model composition algorithm and how it can improve the quality of the predictions and candidates (see section 9.3).
7.2.4 Model Revision

In an idealized setting, where we can rely on a complete constraint solver, the GDE core of the Generalized Diagnosis Engine can come up with three possible classes of results:

- It may find a complete consistent assignment of values for variables, which represents a solution of the generalized diagnostic problem (as shown in the previous section).
- A set of diagnostic candidates may be reported, some of which may contain only user-defined assumptions. Relaxing these assumptions provides a good chance of discovering a solution within the extended forward space.
- If, however, all (plausible) candidates contain closed-world assumptions, this proves that there is no solution within the extended forward space.

Taking a closer look at the first case, we find that it is a rather strong requirement to generate complete value assignments. As there are usually multiple consistent ones, it can be an advantage to represent these classes of solutions as incomplete assignments. However, employing an incomplete constraint solver might lead to missing conflicts, but in the context of a GDE-style diagnostic engine this means at most the failure to exclude some candidates. By design, GDE will never miss candidates, if we take the perspective that consistency stands for all candidates being possible.

The usual starting point for finding a backward completion is the refutation of a local closed-world assumption from an intermediate diagnostic candidate. We are looking for a complement to the generated forward completion to form a potential solution. In the following, we present an algorithmic approach for this task. Section 7.2.4.1 outlines the basic idea, while section 7.2.4.2 presents the incremental completion algorithm that is also used for forward completion. Section 7.2.4.3 discusses its application in backward search and in section 7.2.4.4 some considerations on minimality of backward completions can be found.

7.2.4.1 Idea: Search for Structural Augmentations from a Refuted Local CWA

For an example from the water treatment domain, let's assume we have found the predicted concentration of dissolved iron in a particular water layer to be in conflict with other observations. In the generated conflict, and therefore also as part of the intermediate candidate, the associated local closed-world assumption \( \text{CWA}(\text{DissolvedIronX.concentration}) \) is collected.

As mentioned earlier, we take the domain theory as closed and under this condition, the negation of a closed-world assumption entails a limited set of structural completions: Local closed-world assumptions of type \( \text{CWA_qu} \) are defined as excluding additional influences on quantities. The domain theory contains a finite set of behavior constituent types that can possibly create such an influence as part of their quantity effects - with a given mapping. By considering the quantity role and the type of the object it is associated with, usually only a few behavior constituent types are possible candidates. For refuting \( \text{CWA}(\text{DissolvedIronX.concentration}) \), only instances of behavior constituent types that would create an influence to the concentration quantity of an object of type DissolvedIron have to be considered. This might be transport processes or \text{IronRedissolving} (see section 5.3.3 for the definition).

For any chosen behavior constituent type, the object a quantity is associated with has to be bound to a specific object role, which will in turn restrict the binding of other roles. Furthermore, at least one of the roles occurring in the structural conditions of the behavior constituent type has to be filled with a "new" structural element - otherwise, the mapping would already have been found in the construction of the extended forward space (and, thus, be subsumed under the CWA). We are looking for \textit{additions} to the structure that will instantiate a new behavior constituent. In our example, if there is already a solid layer...
below the given water layer, one possible addition is the introduction of SolidIron in this solid layer - so that the resulting configuration would give rise to an instance of type IronRedissolving, creating the required influence to DissolvedIronX.concentration.

To determine whether this instance really is activated, we propose a test cycle including structural completion from the augmented structure, constraint net generation and prediction. That is, basically, we generate hypotheses about extended structures, so that their associated extended forward space could contain valid solutions. Note that in a test cycle, the closed-world assumptions have to be newly generated, since they refer to a different extended forward space. Of course, all additions have to obey the restrictions imposed on the structure by the use of relation properties - most notably, some positions can be filled only by a single object as defined by functional or unique relations.

And there is still another important condition to be fulfilled, namely the structural configuration to be added to the original structure has to be introducible (see section 6.1.3). This is an important instrument for controlling the allowable backward completions. In constructing structural additions, we have to take this criterion into account. But configurations that are not introducible might still be helpful, as they might be "creatable" from other behavior constituents as structural effects - i.e. they fall into the category of EffectElement but are still part of the backward completion (see figure 7.1 for an illustration). Similarly to finding a behavior constituent (and its structural conditions) to create an additional influence, we can look for behavior constituents creating a structural element needed for another behavior constituent. Thus, the construction of valid backward completions takes the form of a search for an introducible configuration.

So far this section has been referring only to local closed-world assumptions associated with quantities (CWA\textsubscript{aq}). There is, however, the second kind of local CWAs, resulting from resolving the implications for a structural effect element (CWA\textsubscript{spe}, see section 7.2.2.2). If such an assumption is refuted, i.e. there is no support for the element, we first check the element itself for introducibility, which would be the easiest solution: The element is part of the backward completion, rather than an effect of a behavior constituent. In all other cases, we proceed by searching for additional support as outlined above: the element has to be creatable as the structural effect of an additional behavior constituent instance to be discovered.

Additional examples for revisions in the form of structural augmentations found by backward completions can be seen from the application examples in section 8, most notably in the deep search for sedimental iron in the example form section 2 (see section 8.1.1).

7.2.4.2 An Incremental Structural Completion Algorithm

When initially constructing the extended forward space, we have to exhaustively generate every possible mapping of a behavior constituent type's structural conditions to existing structural elements. This can be achieved by several different methods, including generic graph-grammar based approaches (one of the early implementations of the model composition module was based on such an algorithm). However, in creating the respective structural effects, one faces the problem of checking for necessary unifications with existing structural elements, i.e. even object roles from the structural effects could be bound to existing objects.

We have designed a recursive algorithm that incrementally constructs mappings for behavior constituent types to existing or new structural elements - and find that minimal changes in control allow its use for both forward and backward completion! With the incremental approach, even partial mappings from pre-binding some roles for creating an influence or achieving a particular structural effect can be used. In the following, we will present the core of the structural completion algorithm, leaving aside most optimizations.
As data structures, we assume that a **Structure** can store a set of **StructuralElements**, i.e. objects and relation tuples. A behavior constituent type possesses a **StructuralSpecification**, consisting of **ElementSpecifications** for both structural conditions and structural effects - they are distinguished by using the functions **isCondition** and **isEffect**. Finally, a **Mapping** is a set of Bindings of **ElementSpecifications** to **StructuralElements** - and it is associated with a given **StructuralSpecification**. Thus, it can both be checked for completeness w.r.t. to the **StructuralSpecification** (function **isComplete**), and free element specifications can be chosen from it (function **chooseFreeElemSpec**). Note the difference to the concept of mappings in the formal definitions of the semantics (section 7.2.1.2): here also relation tuple specifications are mapped.

A special design decision is to store the definition of a "new" object or relation tuple also in a **StructuralElement**. It can be added to the Structure later. Now, the idea is to create all complete mappings (to existing or new elements) from a given partial mapping as follows:

```java
completeMappings(partialMapping: Mapping, structure: Structure, mode: Boolean) : set of Mapping
    if isComplete(partialMapping) and isValid(partialMapping, mode) then return {partialMapping}
    else
        result := ∅
        elemSpec := chooseFreeElemSpec(partialMapping)
        for each element from possibleElements(elemSpec, partialMapping, structure, mode)
            extendedMapping = partialMapping ∪ createBinding(elemSpec, element)
            result := result ∪ completeMappings(extendedMapping, structure, mode)
        return result
```

The function **possibleElements** selects all those structural elements from the given structure that can be bound to the element specification provided. Depending on the mode (flag **mode** - forward or backward) and whether the element specification belongs to the structural conditions or the structural effects also a new element can be created and added to the set (function **createNewElement**):

```java
    result := ∅
    for each element from matchingElements(elemSpec, partialMapping, structure)
        if fulfillsRestrictions(element, elemSpec, partialMapping) then result := result ∪ element
        if isEffect(elemSpec) or (mode = backward) then
            result := result ∪ createNewElement(elemSpec, structure, mode)
    return result
```

First, all elements that could possibly match the isolated element specification are retrieved from the structure (function **matchingElements**) - here, it is important to make use of existing bindings from the partial mapping, since the connection structure between objects and relation tuples will be the decisive focus in searching for matching elements. Then, each element is checked against any structural restrictions from the existing partial mapping (of course, this can be integrated with the previous step). Alternatively, mappings can be filtered after completion (function **isValid**, see **completeMappings** above). Then, a new element is created if the specification belongs to the structural effects (**isEffect(elemSpec)**) or if we are in "backward mode", i.e. we are looking for additional elements in the structural conditions. Note that **createNewElement** also takes the structure as a parameter and, thus, can check whether the structural restrictions allow the creation of a new element with the given specifications - and return the empty set otherwise.
In this way, all valid completions of mappings are created - the order of choosing elements from the structural specification is relevant only for performance purposes. When starting with an empty mapping, i.e. without bindings, but with the mapping already associated with the StructuralSpecification of a given behavior constituent type, then the "forward mode" will find all configurations underlying behavior constituent instances - and the necessary additions for the structural effects ("augmentations"). Iterating over all behavior constituent types and additionally looping until no more ("forward") structural additions can be found constructs the extended forward space. The final set of all mappings found in this way can be used to create the constraint net. In the simplest form, this can be written as:

```
completeForward(bcTypes: set of BCType, structure: Structure) : set of Mapping
repeat
    augmented := false
    result := ∅
    for each bcType from bcTypes
        emptyMapping := createEmptyMapping(getStructuralSpecification(bcType))
        mappings := completeMappings(emptyMapping, structure, forward)
        result := result ∪ mappings
        for each mapping from mappings
            augmentation := getAugmentation(mapping)
            if notEmpty(augmentation) then
                augment(structure, augmentation)
                augmented := true
    until (augmented = false)
return result
```

Obviously, throwing away all mappings and starting over after each augmentation cycle yields correct results (since all mappings will be discovered again in the augmented structure), but is terribly ineffective. Besides, all mappings that are new in the next cycle will have to involve elements created in the previous cycle, so the algorithm can be optimized by creating all possible partial mappings to the newly created objects instead of empty ones. These improvements are not shown here.

### 7.2.4.3 Searching for Backward Completions

Backward completion will start with a specific local closed-world assumption. For local CWAs of the class CWA\text{qua}, the first step is to identify the object the quantity is associated with and - based on object type and quantity role - will select behavior constituent types that contain an influence on such a quantity in their quantity effects. For local CWAs of type CWA\text{pe}, the respective object type will have to be found in the structural effects. In each case, a partial mapping for a selected behavior constituent type will be constructed that will bind the respective object to the required role - and the structural completion algorithm will be started again, but this time in "backward mode".

The final validity check (see first line of function completeMappings above) is mode dependent in that for the backward mode, mappings are accepted only if their structural conditions contains at least one "new" element - otherwise, the behavior constituent instance would already be included in the extended forward space - and, thus, its influences are already subsumed in the closed-world assumption to be retracted. Regarding the criterion of introducibility, we can distinguish the following cases:

- Objects of a given type or relations tuples of a certain relation are never introducible (cannot be contained in any configuration marked introducible) and do not appear in structural effects. We mark these as non-creatable.
• Elements of a given type can appear in structural effects, but are not introducible themselves. We mark these creatable.

• Elements appear in introducible configurations. In the simplest case, introducibility is determined solely based on the type of an element, for instance by stating that objects of a given type are introducible in any configurations. However, in general we want to be able to judge complete augmentations of whether they make sense (e.g. by distinguishing "where" an object can be introduced in the system - as defined by a locating relation). Thus, we mark these elements as potentially introducible.

In constructing a mapping, new elements, as introduced by createNewElement, can be checked for their introducibility or creatability right away. Thus, non-creatable elements are not considered for mappings at all. Creatable and potentially introducible ones will have to be checked in the context of the complete augmentation at the end:

• An introducible augmentation points to a possible solution - it will be a candidate for a test cycle, i.e. it can be added to the structure and the augmented structure be used as the new initial situation description for searching for a solution by prediction as outlined above.

• An augmentation containing creatable elements can be the basis for further backward search, by trying to find behavior constituents creating it. This is done by binding the object or relation tuple to a structural effect element and using the resulting partial mapping as input for the completion algorithm.

Potentially, there can be a high branching factor in the backward search, and finding minimal combinations of introducible configurations to bring about the required change to the structure can be computationally expensive. However, in many cases, the user has additional information not contained in the model and is in a better position to judge the plausibility of a behavior constituent becoming active. Therefore, G*DE provides an option for interactive searching, in the form of presenting behavior constituents including mappings for a single step (creating an additional influence or creating a non-introducible element) and letting the user choose the path to pursue. The two main functions supporting this are shown in the following. The first one is

\[
\text{findAdditionalInfluence(bcTypes: set of BCType, structure: Structure, object: Object, quantityRole: String) : set of Mapping}
\]

result := ∅
for each bcType from bcTypes
  objSpecs := getInfluencedObjectSpecifications(bcType, object, quantityRole)
  for each objSpec from objSpecs
    mapping := createEmptyMapping(getStructuralSpecification(bcType))
    mapping := mapping ∪ createBinding(objSpec, object)
    result := result ∪ completeMappings(mapping, structure, backward)

return result

The function findAdditionalInfluence produces all (complete) mappings for behavior constituents that create an additional influence on the quantity associated to the given object under the given quantityRole. Hence, it identifies all ways to refute the local closed-world assumption for the specified quantity. However, all influences already known to target the respective quantity are to be disregarded - the task is to find an additional one. This is ensured by completeMappings in the backward mode: only mappings that contain "new" elements in their structural conditions are returned (isValid checks this condition, see above).
The helper function `getInfluencedObjectSpecifications` retrieves all object specifications of a given behavior constituent type (from both structural conditions and structural effects) that can be bound to the given object, so that an influence from the quantity effects targets the quantity with the respective quantityRole. This is not a sophisticated algorithm, except for smart indexing of the behavior constituent definition. All other required functions are known from the pseudo-code fragments above. The central idea is to create a suitable partial mapping and to find all completions using the algorithm outlined above.

The second function looks for additional support for a given structural element, i.e. it detects mappings that define behavior constituent instances with the element in its structural effects. Again, we are interested only in additional behavior constituents that are not already part of the extended forward space:

```plaintext
findAdditionalSupport(bcTypes: set of BCType, structure: Structure, 
    element: StructuralElement) : set of Mapping

    result := ∅
    for each bcType from bcTypes
        elemSpecs = getStructuralEffectsSpecifications(bcType, element)
        for each elemSpec from elemSpecs
            mapping := createEmptyMapping(getStructuralSpecification(bcType))
            mapping := mapping ∪ createBinding(elemSpec, element)
            result := result ∪ completeMappings(mapping, structure, backward)
    return result
```

Partial mappings are created for all behavior constituents with a matching structural effects specification (retrieved by `getStructuralEffectsSpecifications`) and all possible completions are collected in the `result` set. Using these functions, interactive search can be conducted or different automated search strategies be implemented.

For the current G'DE prototype, a "brute force" algorithm for the overall backward search, finding all supports for all creatable elements from intermediate search results has been implemented (not shown here). Currently, this is the only option besides the interactive approach, which relies on the user to define the search path. The high branching factor in the backward search usually causes the highest computation costs in comparison to the complete cycle of model composition, influence resolution, model transformation, prediction, conflict detection and candidate generation. Certainly, future versions will incorporate more sophisticated search modules with focusing strategies or application-specific heuristics.

### Minimality of Backward Completions

As stated in the semantics, valid solutions are required to be minimal w. r. t. the set of "introduced" structural elements (see section 6.2.2). Remember that our search for solutions starts with the carefully constructed extended forward space, and this reflects in our first hypothesis, the global CWA. The global CWA corresponds, by definition, to a minimal backward completion, namely the empty one ($\forall e \neg\text{IntroducedElement}(e)$, see section 7.2.2).

It is only by definite contradiction to this assumption that we start searching for additional structural elements. As a side remark, the appearance of user-defined assumptions together with closed-world assumptions in a G'DE candidate points to multiple possible solutions, as the minimality of the backward extension is not "comparable" to the second requirement of maximal compliance with user-defined assumptions. The addition of elements is treated equally with the retraction of user-defined assumptions from the perspective of validity of solutions, they can, however, be weighted in a later candidate ranking phase.
The crucial point is that we never introduce elements without need. And the algorithmic elements for backward search presented here proceed in a way that guarantees minimality of the retrieved configurations ("structural augmentations") - locally. But be aware that the global requirement for minimality might have to be checked for the complete constructed augmentation, as there is the possibility of locally constructed support to be subsumed by another search path: imagine that additional influences on quantities A and B have to be found. There are two minimal augmentations, A₁ and A₂, supporting an influence on A and one (B₁) for an influence on B. However, not all combinations by union might be minimal, as B₁ could subsume A₁ (B₁ ⊇ A₁), so that A₂ ∪ B₁ is not a minimal augmentation in the given situation!

While we can easily establish a final check for the minimality of augmentation candidates retrieved, it is much more difficult to find valid focusing principles for the searching algorithm. See also the discussion on candidate ranking in section 6.3, where we pointed out that cardinality of augmentations has to be treated with caution as well, since a single "deeper" cause could be responsible for many intermediate search results.

### 7.2.5 Optional Model Transformations

At various stages of the described reasoning process, intermediate results can be interpreted as (more or less implicit) behavior models - and this allows us to apply automated model transformations to them. Model transformations play an important role in G²DE for many practical purposes:

- The requirements for the constraint solver can be relaxed by syntactical transformations, such as rewrite rules. The prototypical implementation employs a module for constraint type normalization, which both replaces multiple aliases of constraint types and provides a breakdown of high-arity constraints into standard ternary ones (see section 7.2.5.1).
- The size and complexity of the model can be reduced by abstractions, simplifications or approximations (see section 7.2.5.2). This can achieve significant efficiency gains, since some operations in prediction and candidate generation are known to be highly dependent on the number of variables or the degree of connectivity in the model.
- Finally, we employ two optional model enhancements, by automatically introducing derivatives (and all deducible constraints for them) and/or deviation models (see section 7.2.5.3) into an existing constraint network. The automatic generation of derivatives is an alternative to the concept of structured quantities, as used in QPT, where each quantity also holds a qualitative abstraction of its derivative. Deviation models, on the other hand, provide a compact way of carrying out comparative analysis - i. e. checking differences to a partially specified reference behavior for consistency. Note that the definition of integrative influence types (see section 5.3.3.6) relies on the derivative enhancement of the underlying models.

Before considering the different classes of model transformations, we give an overview of potential points of integration for transformation modules into the design of the Generalized Diagnosis Engine. Each of these has some unique advantages, while we adhere to the general principle of supplying detailed model fragments in the domain theory (the "model library") that can be adapted to specific tasks by automated transformations.

- Individual behavior constituent types might be transformed, based on criteria specific to the task or the scenario. Examples are matching a given time-scale or the decision of whether to take certain phenomena into account at all ([Forbus 1984] uses "consider" statements for this). This is especially powerful, if the modeler supplies additional information. Imagine a simple model of a chemical...
reaction for the typical temperature range, but more sophisticated ones for extreme values. A special case of this is the pre-selection of behavior constituent types that will be considered for instantiation.

- The XQID generated by model composition (see section 7.2.1) allows for the detection of feedback loops that play an important role in time-scale abstraction, but also several other features of the model can be detected by syntactical analysis in this representation (see [Heller/Struss 1996] for a generic graph-grammar-based transformation approach on XQIDs). We will discuss the impact of time-scale abstractions in some of the examples in section 8.

- For other purposes, the resolved model, i.e. a pure constraint network, is better suited as the basis for transformations. This is where the system architecture diagram for the prototype (figure 7.12) places them - but see also section 7.3.2 for the flexibility in system design. We successfully applied derivative and deviation enhancements to constraint networks as obtained by influence resolution. This simple representation is also suitable for technical rewriting - constraint type normalization is done here, as well.

7.2.5.1 Rewrite Rules

By "rewrite rules" we denote manipulations of the representation of the behavior model that are local in nature (usually applying to a single element, e.g. constraint instance) and that do not change the interpretation of the model in terms of solutions.

In the implemented prototype, we substitute certain aliases for constraint types. One of the advantages is that the user can use meaningful names in different contexts, such as EQUAL for EQUIVALENT (for boolean values), or SUM-1 for EQUAL when applied to a single numerical influence being summed up. The latter example also suggests that some generated constraints, e.g. resolution constraints, can apply a simple naming scheme (SUM-0, SUM-1, SUM-2, ...), without paying attention to special cases.

Another part of constraint type normalization is the breakdown of high-arity constraints. Some associative constraints, like AND, OR, or SUM with an arbitrary number of parameters, can be separated into series of ternary constraints, e.g.

\[\text{AND-n}(\text{res}, a_1, a_2, a_3, ..., a_n) \quad (\text{"res} = a_1 \land ... \land a_n)\]

becomes

\[
\begin{align*}
\text{AND}(\text{res}, a_1, h_1) \\
\text{AND}(h_1, a_2, h_2) \\
\text{AND}(h_2, a_3, h_3) \\
... \\
\text{AND}(h_{n-2}, a_{n-1}, a_n)
\end{align*}
\]

with new helper variables \(h_1\) through \(h_{n-2}\). A more sophisticated technique is described in [Rossi et al. 1990], which allows for many cases the construction of an equivalent constraint net consisting exclusively of binary constraints. Using this kind of transformation reduces the requirements for the constraint solver.

7.2.5.2 Abstractions, Simplifications and Approximations

A classification of useful model transformations can be found in [Struss 1992]. In this work, a relational perspective towards behavior models is taken, i.e. it regards the set of (complete) variable value assignments allowed by the model and its change through a model transformation, so that representation-independent features can be studied. This leads to the following classes:
• **Abstractions:** An abstraction of a model neglects some distinctions made in the original model. This is formalized with the help of representational transformations (see [Struss 1992]), the most common one being the reduction of the granularity of variable domains, e. g. from numerical values to intervals. In this way, each solution of the abstracted model represents a class of solutions of the original model. [Sachenbacher/Struss 2000] presents an interesting approach to automatically determine the minimal distinctions required for a given reasoning task.

• **Simplifications:** A simplification is a model transformation that preserves validity only under a given simplification assumption. E. g. one can rule out high temperature values for a given scenario and therefore exclude the activity of some behavior constituents. By not including these into the composed model at all, the model is simplified, but relies on the simplification assumption (low to normal temperatures) to be valid.

• **Approximations:** Replacing a model with one that is "close" to the original one, but of a simpler form. A typical example is linearizing functional dependencies, so that each solution for the approximated model is close - according to an appropriate metric - to a solution of the original one. "Time-scale abstraction" (see [Kuipers 1987], [Iwasaki 1992]) can be seen as an approximation - it takes a much more abstract perspective to regard it as a proper abstraction.

In the design of G’DE, model transformations can be easily integrated. However, the current prototype contains only experimental modules for XQID transformations as defined in [Heller/Struss 1996].

#### 7.2.5.3 Model Enhancements

Most of the application examples in this thesis rely on model enhancements. This refers to a technique of exploiting the algebraic structure of the model. It turns out that the use of algebraic constraints carries more information than the set of allowable values, so that additional variables and dependencies can be introduced.

For instance, we can automatically introduce derivatives for variables - and determine many dependencies by looking at the dependencies of the original variables. E. g. the constraint

\[
\text{ADD}(a, b, c) \quad \text{("a = b + c")}
\]

can serve as a template for the constraint

\[
\text{ADD}(da, db, dc) \quad \text{("da = db + dc")}
\]

where \( da \) is the derivative of \( a \), \( db \) of \( b \) and \( dc \) of \( c \).

A similar enhancement is the introduction of deviation models. By explicitly representing the deviation of a quantity from an (implicit) reference value, one can reason about the propagation of these deviations. For a variable, \( x \), there is a reference value (that does not need to be known), \( x_{\text{ref}} \), and the actual value, \( x_{\text{act}} \). The deviation, \( \Delta x \), is defined by

\[
\Delta x = x_{\text{act}} - x_{\text{ref}}
\]

Frequently, one uses only a qualitative value for \( \Delta x \):

\[
\Delta x = \text{sgn}(x_{\text{act}} - x_{\text{ref}}) \in \{-1, 0, 1\}
\]

Now, from an additive constraint, \( \text{ADD}(a, b, c) \), like in the example above, one can also see that

\[
\text{ADD}(\Delta a, \Delta b, \Delta c) \quad \text{("\( \Delta a = \Delta b + \Delta c \"\))}
\]
With more sophisticated rules, one can also enhance multiplicative and other constraints forms. For automated construction of deviation models from a system theoretic perspective, refer to [de Jong/van Raalte 1999].

7.3 System Architecture of G^DE

In this section, the overall system architecture of the Generalized Diagnosis Engine, G^DE, is presented. First, the basic architectural elements are outlined and related to the algorithmic design presented in the previous section. In section 7.3.1 the current prototype is examined and section 7.3.2 discusses how the implementation is designed for configurability and flexibility.

Figure 7.11 depicts the architecture of G^DE on an abstract level. A Domain Theory Editor (upper left corner) enables the user to design and modify the domain theory, potentially with a graphical user interface. A Scenario Editor allows the creation of a situation description, which is decomposed into the system structure and quantity specifications. In the initial system structure, all objects and relation tuples appearing in the situation description are collected, while the quantity specifications contain all user-defined quantity value assignments as well as the basic assignments $\text{element}.\text{exists} = \text{true}$ for each element in the initial system structure. Quantity specifications can contain assumptions and especially the assignments to $\text{exists}$ variables can carry user-defined assumptions supporting structural configurations.

Structure Completion, based on the algorithm presented in section 7.2.4.2, constructs the extended forward space from the domain theory and the system structure. In the course of completion, the system structure is extended. As a result, a set of behavior constituent instances (in the form of "mappings") is passed to Constraint Net Generation, which handles the encoding of all conditions and effects into the conditional structure (see section 7.2.1), as well as influence resolution (section 7.2.2) and optional transformations (section 7.2.5). The resulting constraint net is basically an encoding of the extended forward space including the complete conditional structure for reasoning about existence and activation.

A GDE module then uses this constraint net ("SD") plus the quantity specifications ("OBS") and proceeds with prediction, conflict detection and diagnostic candidate generation. Note that the user-defined assumptions associated with certain quantity specifications have to be taken into account here. This achieves the consistency check described in section 7.2.3. Candidates that contain closed-world assumptions are handled in the Model Revision module. The fundamental backward search algorithm presented in section 7.2.4 relies on the Structure Completion once more, but the backward arrow indicates also the generate-and-test loop for the completed structure.

The Scenario Editor is also intended for all user feedback by displaying the augmented structure, the computed quantity values, and diagnostic candidates. It can also be used for interactively selecting search paths for model revision.
7.3.1 Design of the Current Prototype

Our prototype is implemented in the form of configurable modules, mostly in the form of Java applications. All data structures to be passed between modules are defined in the form of XML documents ([XML 1998]), more precisely there is an XML Schema ([XML Schema 2000]) for each exchange format. This applies to domain theories, system structures, quantity specifications (including assumptions), behavior constituent instances, XQIDs, constraint nets, and diagnostic candidate sets. The use of XML provides maximum flexibility w. r. t. the processing of the data. In fact, many experimental modules or early implementations have been realized as generic transformations of XML documents, using the defined standard called "XSL transformations" ([XSLT 1999]), which has the advantage of freely available implementations of XSLT processors.

Three modules are based on commercially available tools, namely the diagram editor Visio ([http://www.visio.com]) for the graphical user interfaces and the flexible GDE implementation from OCC'M Software ([http://www.occm.de]), called RAZ'R Runtime System. Figure 7.12 presents a more detailed view of the G'DE prototype.
The *Structure Completion* module (upper right) is responsible for both forward completion and backward search (the latter corresponding to *Model Revision*, cf. figure 7.11), indicated as different control modules for a common structure composer, as described in section 7.2.4. The *Constraint Net Generation* is distributed in three submodules, in analogy to the steps described in sections 7.2.1, 7.2.2, and 7.2.5: First, the behavior constituent instances are collected into an Extended Qualitative Influence Diagram (XQID), which is passed to the resolver. There, the summary constraints and the local closed-world assumptions are created for each set of influences on a common target quantity. Finally, an optional set of transformers can simplify, enhance or normalize the resulting Extended Constraint Net (XCNet).
The GDE core is encapsulated with converters for translating the XCNet into the proprietary system description (SD) format, which is basically a set of (pseudo) components plus definitions of constraint types used. As mentioned before, the RAZ'R system has the advantage of being open to arbitrary defined constraint types over finite domains. Quantity specifications are converted into "observations" - and prediction results back into quantity specifications for feedback purposes. The "QS-to-OBS" converter has the additional task of "pre-feeding" the ATMS (creating atoms and justifications) of the RAZ'R system with the assumptions encountered in the initial quantity specifications. Finally, the candidate extractor polls the GDE core for diagnostic candidates that can even be created incrementally as prediction progresses.

The user interfaces are located to the left of figure 7.12. Visio diagrams for the domain theory and the situation description are created from a given set of graphical elements, called stencil, and with some support in the form of scripts and macros. An example screenshot is shown in figure 7.13 with the stencil to the left. Actually, most of the illustrations in section 8, as well all of the examples for object hierarchies, relation definitions, quantity definitions and behavior constituent types in section 5 are exported directly from the domain theory editor.

These diagrams are automatically analyzed by the modules DTAnalysis, SSAnalysis, and QSAnalysis, and stored in the form of XML files complying to the respective schemas. All three modules are, in fact, realized as different scripts for a general tool (GraphicalXML) that was developed for the retrieval of XML descriptions from graphical connection structures.

Figure 7.13: Screenshot of Visio interface (editing a behavior constituent)
Graphical user feedback in the form of the display components for structure layout (SSDisplay, see figure 7.12 again) and quantity querying (QSDisplay) is not implemented yet. Also, a simple HTML-based candidate display is currently used. These are perspectives for future development (see section 9.3).

7.3.2 Configurability and Flexibility of the G’DE Implementation

The design of the Generalized Diagnosis Engine, and in particular of the current prototype is targeted towards maximum flexibility and the configurability of the modules allows the construction of specialized problem solvers.

Firstly, the set of constraint and influence types that can be used in the modeling language is open to additions, as mentioned before. Besides a required basic set (used for the construction of the conditional structure) and some restrictions (e. g. the necessity to provide a conditional form of each constraint to be used in quantity effects), one can use any set of constructs, as the algorithms passes them to the predictor. In our case, the predictor can be supplied with user-defined constraint types. But note that model transformations might rely on a particular constraint set to be analyzed.

Secondly, the modularized architecture allows the integration of new modules. For instance, more powerful model transformations - potentially tailored for specific constraint sets - can be employed. In figure 7.12, a variable set of transformers is depicted that can be used after influence resolution. With the current implementation, we supply two optional model enhancements (deviation models and derivatives), constraint normalization, and a simple abstraction module, see section 7.2.5 for an overview and perspectives for additional transformations. Of course, the accessible XML data formats allows the easy integration of model transformations also for other intermediate result, e. g. the Extended Qualitative Influence Diagram (XQID) can be analyzed before being passed to the reformer, or individual behavior constituent types can be transformed. Also pre-computing simplifications of the system structure is possible.

Thirdly, specific modules can be replaced, e. g. for improved performance in special cases. A subclass of models allowing significant simplifications is behavior constituents without structural effects, which covers many QPT examples (and all of PDE, see section 9.1.4). In these cases, one can rely on a much simpler composition of the conditional structure, as only quantity conditions and activity have to be taken into account. Another example is a more powerful predictor or diagnosis engine, which might handle numerical calculations or even simulation. We have taken care, not to restrict the modeling paradigm to purely qualitative information, although it is very common in consistency-based diagnosis to do so. In principle, composition and consistency check for partial differential equations is conceivable, although the high requirements for focused conflict detection are usually an obstacle (it is certainly unacceptable to always have all solutions of an equation system rely on all model assumptions). Also see the discussion of the integration of temporal aspects into G’DE in section 9.3.

Finally, reconfiguration of the existing modules allows for using different input sources as well as providing advanced display capabilities. Input sources are not limited to improved editors, also algorithmic construction of system structures to be tested or automated generation of quantity assignments from monitoring equipment are conceivable. For the output of results, we have envisioned graphical feedback within the scenario editor (see section 9.3 for perspectives) and interactive control of the diagnostic process, but also post-processing of results and higher-level generate-and-test control is possible.
8 Application Examples

In this section, we provide a set of extended examples from three different domains, namely water treatment (section 8.1), medicine (section 8.2), and electrical circuits (section 8.3), before discussing general applicability (section 8.4). Each of the examples has a different focus and tries to highlight some of the specific possibilities and benefits inherent in the presented approach and, more specifically, in using the Generalized Diagnosis Engine. This serves at the same time as proof of applicability, as an illustration of the inner workings of the diagnostic engine and as a simple tutorial on problem solving using G'DE. We hope that modeling techniques and applications can be learned from the extensively illustrated examples.

Note that all graphical representations of ontologies, quantity associations, behavior constituent types and scenario structures are directly exported from the respective editing tools in G'DE (the domain theory editor and the scenario editor, see section 7.3). The graphical modeling elements facilitate the study of the examples.

Quantity types have been omitted throughout the example section. In most cases, real valued quantities make sense, but the current prototype is restricted to finite qualitative variable domains, mostly just signs, which proved sufficient for all presented examples. See, for example, the detailed diagrams of how values and conflicts are derived (figures 8.13 and 8.24) by using qualitative derivatives only.
8.1 Water Treatment Examples

First, we would like to revisit the motivating example that was introduced in section 2 and a related one from the domain of water treatment and hydro-ecology. We will present the domain theory and the situation description as entered into $G^\text{DE}$, as well as some diagrams of how $G^\text{DE}$ internally generates predictions, conflicts, intermediate candidates, and structural enhancements to the initial scenario.

The focus of the first example (section 8.1.1) is a deep search tree for model revision, spanning multiple behavior constituents, until an introducible augmentation is found. The second example (section 8.1.2) features a more complex initial model for the detection of a primary conflict.

8.1.1 Sedimental Iron Causes Bad Taste

As input for $G^\text{DE}$ we need a domain theory and a situation description. Figure 8.1 represents the object type hierarchy used for the water treatment examples. Part of it was already used in section 5.3 to illustrate the modeling primitives. The hierarchy of spatial locators (to the left of the figure) also makes use of multiple inheritance, so that e. g. a WaterLayer is both a Layer (so it can participate in the spatial relation below, see figure 8.2 below) and a LiquidLocator (enabling it to be connected to a Pump and to contain dissolved and suspended substances, as well as Colloids). See the relation definitions below for the rationale behind this hierarchical organization.

Next, the relations are defined as shown in figure 8.2. Again, some of them have already been discussed in section 5.3. Note that the different substance classes have different associated unique relations to locate them in a liquid or solid locator, respectively. A future extension will allow a single relation to be "unique" or "functional" for object types that are more specific than their defined parameter type - thus, a relation located-at could be unique for DissolvedIron, which is a DissolvedSubstance and therefore a
spatial entity (see hierarchy above). See the perspectives section (9.3) for a discussion of the planned extensions. Here, a viable workaround is used.

![Figure 8.2: Relation definitions for the water treatment examples](image)

Quantity associations for the various object types can be seen from figure 8.3. The taste of DrinkingWater is assumed as a linear variable so that deterioration can be assessed quantitatively or qualitatively. The modeling technique of associating a concentration quantity with a "uniquely" located substance to represent the concentration of the substance in the location has been discussed in section 5.3.1 already.

![Figure 8.3: Quantity associations for the water treatment examples](image)

Now, we will present a set of behavior constituent types from the water treatment domain theory that describe phenomena related to iron.

![Figure 8.4: Behavior constituent type "Iron Redissolving"](image)
First, figure 8.4 is the behavior constituent type "Iron Redisolving" that is well-known from the detailed discussion in section 5.3.3. For details about the chosen functional influences resulting from a partial time-scale abstraction, see section 8.2.1.

Figure 8.5 describes how dissolved iron can ascend from the lower layers of a water body to the higher ones. In effect, a difference in concentrations will be leveled out. This is a variant of the transport prototype that can be seen in the next behavior constituent.

![Figure 8.5: Behavior constituent type "Iron Ascending"

The transport of dissolved iron along with a water flow through a pump is defined in figure 8.6. The concentration difference as well as the actual flow through the pump drive the assimilation in iron concentrations. In equilibrium, both concentrations are equal again. Note that the concentration of the source remains unchanged, since the water will be not be "thinned out".

![Figure 8.6: Behavior constituent type "Iron Transport"

In figure 8.7, the behavior constituent for iron perception is depicted. Any dissolved iron in the drinking water supply is assumed to possess a negative influence on the taste of the drinking water. The distinction of taste variants or "flavors" is irrelevant for the scenarios presented here.
There are two more behavior constituent types in our domain theory: Alkalization by calcium carbonate and oxidation are shown in figure 8.8.

Remember the informal description of the example from section 2. We repeat the diagram here (figure 8.9), before attempting a formal encoding as a situation description for the diagnostic engine.
The following (figure 8.10) is the structure of the situation description in terms of the object types and relations defined in the ontology. *Hypolimnion* is a common term for the lower layers of a stratified water body, while *Epilimnion* represents the upper layers. The remainders of the treatment plant after the first tank are skipped as they prove mostly irrelevant for iron concentrations, but see the next example (section 8.1.2) for a more detailed model here.

![Diagram of the structure of the situation description](image)

**Figure 8.10: Formal encoding of the scenario description for the "sedimental iron" example**

As a quantity specification, we add the observation of a decrease in perceived taste of the drinking water plus the default assumption that the flow through the pump is constant. Note that this example, like the majority of the remaining ones relies on the ability of GDE to introduce derivatives of quantities automatically. This feature is part of the automated model transformations described in section 7.2.5. The observations are encoded in a three-valued qualitative domain, \{decreasing, const, increasing\}, for the derivatives of the respective variables:

\[
\frac{d}{dt} D.taste = \text{decreasing} \\
\frac{d}{dt} P1.\text{flow} = \text{const}
\]

Starting situation assessment, we label all substances introducible. Interestingly, this example starts with the equivalent of an "empty model", as no behavior constituents are instantiated for the given scenario. Of course, in a larger application context, more elements of the situation are known, as well as a larger domain theory is provided, so that a number of behavior constituents are occurring - but the part of the composed model with relevance for the drinking water's taste and iron concentrations (which will be determined to be the only plausible cause of the observed deterioration) is still empty. The only interesting step in forward completion is the final influence resolution that will create the important closed-world assumption \(CWA(D.taste)\).

This assumption supports the default for the derivative of the taste \((d/dt \ D.taste = \text{const})\) and it is the only candidate generated by the GDE core from the obvious contradiction with the observations. The focus of this example clearly lies on the backward search for structural revisions, that could be responsible for the undesirable behavior.

A single behavior constituent type features an influence on a taste quantity, namely *IronPerception*, a single instance of which is found to be creatable - for the defined set of introducibles. If we would allow the introduction of a supplies relation tuple, i. e. a secondary source for the same drinking water, then, of course, this could lead to another potential cause of the bad taste.
However, this is not the case and the supposition of dissolved iron in tank T1 is the only choice. For the time being, dissolved iron is deemed a sufficient explanation, and G^DE finds a solution scenario like the one displayed in figure 8.11.

![Figure 8.11: A simple solution to the "metallic taste" scenario](image)

However, we'd like to inquire into the causes of the presence of dissolved iron (\_Iron1) in tank T1 - and so we start a new situation assessment and mark dissolved iron generally as non-introducible. This requires the search for acceptable augmentations to proceed further.

As a side remark, a "test" cycle with the structure from figure 8.11 would reveal a conflict including the newly created CWA(_Iron1.concentration), and its refutation yields exactly the same results in terms of further causes as described in the following - derived in multiple test cycles.

Non-introducible dissolved iron still belongs to the "creatable" elements, so the augmentation mentioned above can be the starting point for further search. In fact, it is still the only intermediate candidate to be found. To support the element, a behavior constituent instance is needed that establishes it as part of its structural effects. Again, G^DE will come up with a single match, namely iron transport through the pump - and the existence of dissolved iron in the epilimnion (unnecessary to mention that we do allow the introduction of a second pump). Still, dissolved iron is not allowed as a final solution anywhere in the system - and so another search step finds iron ascending from the hypolimnion - again proposing (creatable) dissolved iron.

Finally, the search algorithm discovers the redissolving of solid(!) iron from the sediment as a source. Solid iron is still marked as introducible and we are satisfied with the situation hypothesis for now. Note that the search did not require intermediate test cycles to come up with this solution candidate. As mentioned before, restrictions in the set of introducibles result in the discovery of deeper causes for the observed symptoms.

Now, the discovered candidate for a solution to the situation assessment task can be seen from the structure in the following diagram (figure 8.12).
The diagram also shows all instantiated behavior constituents, sometimes in a slightly abbreviated fashion (the helper quantity "difference" is omitted in the iron transport instance, and no quantity conditions are displayed). This is basically a representation of the composed model as generated in a test cycle for the augmented structure. Marking pH generally as a closed quantity (role) yields a perhaps surprising result: the newly created situation hypothesis is still not consistent with the observation! This time, the inconsistency will be discovered by prediction in the somewhat larger model. We will show the propagation of values as generated in the GDE core together with the closed-world assumptions recorded. See figure 8.13 for an attempt of a graphical representation.

The picture might look confusing, but traversing it from top right, where the crucial observation $d/dt D.taste = decreasing$ is located, will help. Values above quantities are observations, while the ones placed below are derived. We have focused on the derivatives only - and for most constraint types involved there is a simple parallel version for the derivatives (see section 7.2.5). Multiple arrows towards a single quantity denote that more than one value is used in their prediction - and all of the associated assumptions are collected. The closed-world assumptions are shown with dashed arrows, where used in a prediction.
The diagram should clarify, how the pH of the Hypolimnion is predicted to be decreasing, which is in contradiction to its own closed-world assumption, setting it's derivative to const, instead. Thus, the single conflict found by the GDE core contains the following assumptions, all forming intermediate candidates:

- CWA(D.taste)
- CWA(_Iron1.concentration)
- CWA(_Iron2.concentration)
- CWA(_Iron3.concentration)
- CWA(_SolidIron1.concentration)
- CWA(Hypolimnion.pH)

The last one is the key to finding further causes of the low pH - which will not be covered here. But compare the result to the informal presentation of the situation assessment from the introduction (figure 8.14).
As a next step, therapy recognition requires that we explicitly state the goals for the described situation. It's not necessary to have tasted water from a tap in Lomba do Sabão during algal blooms to agree that $\frac{d}{dt} \text{D.taste} = \text{increasing}$ is a desirable thing. Keeping the assumption of a constant flow the pump, the diagnostic engine will come up with the same conflict as in the previous step. In fact, the prediction scheme displayed in figure 8.13 is in perfect analogy to the one required here, only that the signs are reversed (increasing swapped with decreasing and vice versa).

Thus, the closed-world assumptions collected point to ways to influence the system in a desirable way. It will turn out that most of these do not represent useful alternatives, as, for instance, we cannot decrease the concentration of solid iron in the sediment (which would, in a way, be an ultimate cure for the problem). But there is simply no means to achieve this - and this is reflected in missing behavior constituents for this purpose in the domain theory. But the behavior constituent types Alkalization and Oxidation (see figure 8.8) are instantiated as potential solutions. The introduction of an oxidation agent is proposed for all locations (LiquidLocators, to be precise), and, in principle, Oxidation could be applied anywhere. It is for economical reasons - dispersion of the oxidation agent is not modeled here - that the only pragmatic solution is to locate it in the tank. For Alkalization there is only a single match, namely to add calcium carbonate to the Hypolimnion and thus raise the pH there.

---

Figure 8.14: Informal interpretation of the situation assessment results (cf. figure 2.2)

Figure 8.15: Informal overview of therapy options in the "sedimental iron" example
Hence, G^DE correctly proposes both options for immediate intervention mentioned in the real-world scenario (see figure 8.15). The other two options mentioned (algaecides and eutrophication protection) are considered preventive measures for future occurrences. We do not discuss the temporal aspects here, nor do we go into the details of modeling algal blooms (see [Heller et al. 1995] for some of the issues involved in creating convincing qualitative models for the phenomenon). We rather continue with a different example, instead.

### 8.1.2 Failing Coagulation

Our next example focuses on a different problem with the preparation of drinking water. For this, we take a closer look at the treatment plant. In particular, the flocculation tank and the sedimentation tank are concerned with the removal of colloidal substances from the raw water (see figure 8.16 for a schematic).

The flocculation tank is continuously stirred and aluminum sulfate is continuously added. This causes the coagulation (flocking) of colloids to larger suspended solids, which will settle down later in the sedimentation tank. The problem with coagulation is the strong dependence on pH values - within a certain limit, larger pH values yield better results and below a pH of 6, aluminum sulfate is mostly ineffective. Before the water enters chlorination, a check for organic colloids is advisable, since they pose the risk of building chloroform compounds that are highly undesirable. For our example, let's assume, a general increase in colloids has been detected after sedimentation.

For the formal representation of the example and its transformation into G^DE input, we can make use of the ontology presented above (figures 8.1 and 8.2) with the minor addition of aluminum sulfate as a dissolved substance. A few additional behavior constituent types will be needed:
Figure 8.17 defines the two behavior constituent types "colloids transport" and "suspended solids transport" - and the obvious similarity between the two suggests that a template mechanism would be desirable to support the creation of analogous types (see perspectives in section 9.3). Both show the transport of substances along with a pumped water flow between two LiquidLocators, which can be tanks or compartments of a water body.

Figure 8.18 describes how the pH of the destination of a pumped flow changes with the pH of the source. Again, in all three behavior constituents, we have applied time-scale abstraction as discussed in section 8.2.1.

The important behavior constituent type "coagulation" (figure 8.19) represents the transformation of colloids into suspended solids in the presence of the particular flocculation agent aluminum sulfate. The influence of pH is modeled as linear (as a factor for the process rate) with a cut-off at 6, where the behavior constituent will become inactive.
The "sedimentation" behavior constituent type (figure 8.20) allows us to model the sedimentation tank as a location with minimal turbulence - so that the quantity condition defining the activation threshold, **LESS-THAN-x**, is triggered. When active, it establishes a negative feedback loop for the suspended solids concentration - higher concentrations will settle faster. The equilibrium (achieved at long term only) is located at zero concentration for suspended solids. We have not modeled sludge production and continuous sludge removal for this example. Obviously, colloids will not settle down.

Next, we present the situation description (figure 8.21) for the example, consisting of two tanks, the first one with the addition of aluminum sulfate and the second one with zero turbulence, thus qualifying as a sedimentation tank. For the sake of simplicity, we have connected the first tank directly to the epilimnion via a pump, thus skipping pretreatment steps. The pumps are working to achieve a constant flow and the original colloids and suspended colloids concentrations as well as the flocculation agent dosage are steady. The crucial observation is the increase of colloids in the sedimentation tank.
Multiple behavior constituent instances will be created for the given scenario:

Figure 8.22 shows a fragment of the composed model, without sedimentation (instantiated for all three locations, but active only for the last) and coagulation (only one instance for the flocculation tank, this
can be seen later on). Again, we have resorted to a slightly simplified representation of behavior constituent instances, omitting helper quantities and quantity conditions. The next diagram (figure 8.23) shows the relevant part of the constraint/influence network generated from the composed model, so that we can take a closer look at conflict generation again. Variable names are qualified with the object name they are associated with. For the coagulation behavior constituent instance, we have left out the influence on suspended solids.

![Figure 8.23: Relevant part of the constraint/influence network generated from the composed model](image)

The following diagram (figure 8.24) shows how the central conflict

\[
\{\text{CWA}(\text{C2.concentration}), \text{CWA}(\text{Tank1.pH}), \text{CWA}(\text{Epilimnion.pH})\}
\]

is derived.

![Figure 8.24: Prediction as constraint propagation in the crucial part of the constraint/influence network](image)

Again, values above quantities denote input, while values below indicate predictions. The diagram can be interpreted best from the far right, starting with the observation of the increase in colloids C3. Once more,
prediction focuses on the derivatives of the quantities involved. The use of the various local closed-world assumptions is indicated.

The diagnostic engine correctly determines that the pH in the epilimnion is decreasing (contrary to the model closure assumption). This points at a helpful situation assessment for the given scenario. Once again, we leave it open how the pH can be influenced. But note that the pH in the sedimentation tank is predicted to fall, as well, which provides another way to confirm the situation assessment from correspondence with actual observations (this would be a measurement proposal, a task beyond the scope of this work).

The conflict derived above is produced identically in a therapy recognition session when entering the goal of reducing the final colloids \((\text{d/dt } C_3.\text{concentration} = \text{decreasing})\). This means that all behavior constituents directly removing colloids (none known) or to increase pH are welcome additions. In the latter category we have alkalization by calcium carbonate defined above (figure 8.8), which will be proposed for any tank by G^DE - and this is what is actually employed in the case of failing coagulation.
8.2 Medical Examples

[Patil 1988] argued that model-based diagnosis had nothing to offer for medical diagnosis. This argument is based on the correct observation that the component paradigm is inadequate for reasoning about physiology, since the simplistic mapping of function to structure does not meet the requirements. We argue that the powerful consistency-based diagnosis approach can be freed from this limitation and the presented process-oriented extensions should prove valuable in the medical domain. Of course, one has to be aware that in health care routine and also in most of medical research we are confronted with a very different form of knowledge and style of reasoning: there is a single type of system with a known range of variations, namely the human body. It has been under observation for several hundred years with varying methods but steeply increasing precision and sophistication and for most kinds of "faults", i. e. illnesses and diseases, literally millions of instances are documented and most possible therapies are well tested and studied.

So, certainly, we do not expect the general practitioner to experiment with a version of G'DE to generate innovative explanations for the symptoms presented by a particular patient and then to start searching for therapeutic options. Rather, the following examples are to be understood as a contribution to medical theory formation and could be helpful in comparing competing hypotheses for consistency and/or observable behavior predictions. Both of our examples model well-known mechanisms and we take pride in elegantly reconstructing all known solutions.

The focus of the example in section 8.2.1 is on diagnosing systems involving feedback loops, while the example in section 8.2.2 highlights the handling of degradation faults.

8.2.1 Renal Artery Stenosis

Vascular hypertension, or "high blood pressure" is a condition with a very high prevalence in western societies. For a small number of cases (about 10%), a specific organic cause can be found - these cases are labeled "secondary" or "symptomatic" hypertension. Of these, we consider a scenario for hypertension caused by a partial blocking of the renal artery (the main blood vessel supplying the kidneys), which throws the renin-angiotensin regulation system off balance. This mechanism is reported to account for about 5% of total hypertension cases, i. e. half of all symptomatic ones ([Pschyrembel 1986, "Nierenarterienstenose"]).

The ontology for the physiological examples (this one and the following one) is shown in figure 8.25. The hormones renin and angiotensin II play a major role in the mentioned regulatory cycle, as the kidney releases renin to increase blood pressure, if it happens to be to low. In the opposite case, i. e. when blood pressure in the kidneys is too high, renin levels are lowered, so that it will decrease again. We lump the rest of the cardiovascular system (blood vessels, heart, and lungs) together as a single "organ", in which renin will result in the production of angiotensin II - a more detailed model would include angiotensin I as an intermediate substance. See below for a discussion of the behavior constituents involved.
Single blood vessels can be included explicitly in the model to connect organs. We distinguish between parts of the high-pressure system (arteries) and parts of the low-pressure system (veins), with pressure transmission being relevant only in the former case. Alternatively, one could explicitly model capillaries and the associated pressure loss, but this is omitted here. Also, the consumption of hormones transported into the cardiovascular system is left out - hence, for the sake of simplicity, in our example hormones are transported only by the veins, i. e. from the producer to the consumer.

Relation properties are mostly unimportant, except that the in relation is to be functional, complete and unique for all leaf types. See section 8.1.2 for a discussion how relations are split into separate relations for all leaf types as a workaround for the uniqueness label being too general. In the domain theory actually used for the example, we have done so, figure 8.25 being a simplification.

As for the quantities representing the behavior of the system, we use the concentration of hormones in a given organ, the blood pressure in an organ and the opening (area) of an individual blood vessel (see figure 8.25, as well).

Figure 8.26 shows one of the behavior constituent types used: renin release within the kidney. The release rate is anti-proportional to the current blood pressure (see the constraint NEG employed). Additionally, we have included a quantity condition representing a certain threshold ("x") for renal blood pressure, above which no renin is produced. In the following, this quantity condition is not shown anymore. Note that renin release is a quick response to low blood pressure (within minutes or hours). See below for a detailed discussion of the time-scale issues of the example.
We will not list the remaining behavior constituent types as they can be easily reconstructed from the instances shown in the composed model (see figure 8.28 below).

Figure 8.27: The initial scenario of the stenosis example

Figure 8.27 depicts the structure of the initial scenario for the stenosis example. A kidney (K) and the rest of the cardiovascular system (CVS) are connected by an afferent vessel (artery A) and an efferent vessel (vein V). All hormones present will be introduced by respective behavior constituents.

The following (figure 8.28) is a diagram of the completed structure and all behavior constituent instances.

Figure 8.28: The composed model of the stenosis example (with all created structural elements and behavior constituent instances)

This forms an adequate representation of the renin-angiotensin system. The blood pressure of the overall system is transmitted via the artery A to the renal system (kidney K). The presented behavior constituent type "renin release" (see figure 8.26) is instantiated for the kidney and the hormone is transported into the outer blood vessels through the vein V. Again, we employ the modeling technique of having single instances of a common object type (Renin) represent "the" concentration of a substance in spatially
distinct regions: objects _R1 and _R2 in their respective locations indicated by the relation in. Renin in the cardiovascular system triggers the production of angiotensin II, which acts as a strong vasoconstrictor, thus increasing the overall blood pressure.

Before we proceed to the description of observations, i.e. quantity values to be included in the initial scenario, we have to discuss an important modeling issue, namely the time-scales of the processes involved. All of the behavior constituent types used so far act on comparable time-scales. We are talking exclusively about medium-term effects in the course of minutes to hours. Section 7.2.5.2 mentions work in automated model transformation, especially on time-scale abstraction (see also [Kuipers 1987], [Iwasaki 1992], [Heller/Struss 1996]). The basic idea behind this technique is that from the perspective of much slower processes, a particular process or mechanism can be seen as acting instantaneously, i.e. without delay. In the case of an equilibrium process, this means that a delay or even an oscillation period before achieving equilibrium can be neglected. Conversely, from the perspective of much faster processes, a particular effect can be regarded as constant. While a discussion of all underlying assumptions is beyond the scope of this thesis, we state that the quality of the approximation increases with the separation of the time-scales involved.

In our example, we observe that the overall blood pressure is steadily increasing on average over months. This means that short-term disturbances, e.g. by exercise, can be disregarded - instead, there is a definite long-term behavior to be explained. Furthermore, it gives us the possibility to model all medium-term effects as instantly achieving equilibrium. The temporal dynamics of renin distribution within the blood stream are of no interest in the given scope, so we can represent the angiotensin level as directly coupled to the renin level (the single "compartment" of the cardiovascular system doesn't allow for spatial distinctions anyway). All of the processes in this example are modeled as the result of such a time-scale abstraction. ([Heller 1995] contains a formal treatment of the conditions and the derivation of bounds on errors committed for some important cases.)

In section 7.2.5 we have also seen how a given model can be enriched by additional variables. In the case of this example, we will have to make use of the automated generation of derivatives for all of the quantities. This is necessary to adequately encode the observation of a long-term increase of blood pressure as

\[ \frac{d}{dt} \text{CVS.blood\_pressure} = \text{decreasing} \]

and for prediction and candidate generation described in the following.

The extended forward space generated from the given scenario has already been visualized (figure 8.28). If we close the model completely, i.e. resolve influences for all object quantities and, thus, create closed-world assumptions for them, the observation leads to an inconsistency. G'DE will detect a single conflict, which is not very surprising, as the complete feedback loop is involved: We see that for constant, since uninfluenced, opening areas for both blood vessels (A.opening and V.opening), the blood pressure within the kidney would rise as well, therefore the renin concentration both inside and outside would go down, as would the angiotensin II level, leading to a single negative influence on the overall blood pressure, CVS.blood\_pressure.

The conflict collects all of the closed-world assumptions in this loop and so we end up with seven single intermediate candidates:

- CWA(CVS.blood\_pressure)
- CWA(A.opening)
- CWA(K.blood\_pressure)
- CWA(_R1.concentration)
Consequently, there will have to be an additional influence on one of the variables. This is the usual pattern for a negative feedback loop moving out of equilibrium and the strength of our approach does not show in the generation of theses candidates, but rather in what to do with them. It turns out that there are no relevant effects releasing renin or angiotensin except the ones already modeled, thus eliminating the candidates $\text{CWA}(_{R1}.\text{concentration})$, $\text{CWA}(_{R2}.\text{concentration})$, and $\text{CWA}(_{A1}.\text{concentration})$ for lack of consistent refutation to be found. While it is obvious that there are a number of effects raising the overall blood pressure ($\text{CVS}.\text{blood\_pressure}$) we have not modeled - and these are, indeed, the most likely ones - we would like to draw attention to the remaining candidates.

The blood pressure within the kidney is hard to influence directly, so we disregard it, as well. Trying to refute $\text{CWA}(\text{V}.\text{opening})$, i.e. influencing the veinal opening area quickly leads to the discovery that a positive influence is needed - which is also very rare, especially as a persistent and continuous one. This leaves us with an influence on the opening of the artery ($\text{A}.\text{opening}$), and this is indeed an interesting possibility, when we consider the process of arteriosclerosis defined in the following figure (8.29):

In the presence of certain toxins (that cause small lesions of the endothelium), high blood pressure causes clogging of arteries. We simply model a negative influence on the opening area. Adding an instance of this behavior constituent type yields a satisfactory solution. Note that the time-scale is comparable to the one of the observations - within months or years, the arteries become narrower ("stenosis" in medical terms) and the renin-angiotensin system increases the blood pressure further. The regulatory response is fast in comparison and its inner dynamics are of no importance. Certainly, a more sophisticated reasoning system could control the reasoning process explicitly by time-scale information (see Perspectives, section 9.3)

Now, what about therapy? Adding the goal of lowering the blood pressure leaves us with a therapy recognition scenario. And there is indeed a treatment for many cases renal artery stenosis, namely balloon angioplasty. A balloon catheter is inserted into the vessel and then pneumatic force is used to widen the diameter and/or flatten deposits on the walls. See figure 8.30 for a model fragment for this action.
Figure 8.30: The "Balloon Angioplasty" behavior constituent type which constitutes the preferred therapy recognition

The focus of this example is not on detailed action modeling, as we do not directly support reasoning about action and change in our approach. We are satisfied that the respective behavior constituent will indeed be found and instantiated - proposing literally the "introduction" of the balloon into the blood vessel to the user. It is important to note that it represents a short-term intervention - rather than a continuous antagonistic process. Certainly, conventional hypertension medication is advisable, as well.

8.2.2 Causes of Proteinuria

Our next example is concerned with the kidney and its blood vessels, as well. Assume a routine screening discovers albumin, a protein, in the urine - the clinical condition is called "proteinuria". For the doctor, this is an alarming sign, if repeatedly confirmed - we will see how to convey this urgency to G'DE, as well.

Expanding the ontology presented above a little, we include albumin and the functional unit of the kidneys responsible for the filtering of extra-cellular liquid, the glomeruli. A quantity is used to encode the percentage of broken glomeruli (microscopic tubes): glomeruli.broken (see figure 8.31).

Figure 8.31: Additions to the physiological ontology (cf. figure 8.25)

The decisive new behavior constituent type is the filtering achieved by the glomeruli. Figure 8.32 depicts this process. The integrity of the glomeruli is the central factor for blocking albumin from passing into the urine - modeled here as a multiplicative constraint setting the passing rate to zero, when there are no broken glomeruli. If we cannot envision another source, this effectively sets the albumin in the urine to zero. After all, this is the purpose of a filtering process.
The scenario then accommodates the additional differentiation of the kidney as follows (see figure 8.33):

Again, we will give a presentation of instantiated structural elements and behavior constituents for the initial scenario, however, we will restrict the diagram to the relevant ones - of course, all of the above behavior constituents belonging to the renin-angiotensin system will be present, as well, but they will not directly interfere with the behavior currently under consideration. Here, the fragment shows only albumin transport and glomerular filtering (see figure 8.34).

Albumin (in contrast to hormones, which are consumed) is moved both through the artery and the vein - and for the healthy case of constant opening areas, a concentration equilibrium will be maintained. The interesting part is the filtering: as said before, the glomeruli are the only path generating urine and, thus, a healthy filtering system (G.broken = 0) will not let any albumin get there.
There are basically two options how to state the observations versus the healthy conditions - raising a contradiction. Firstly, we can state the integrity of the glomeruli (G.broken = 0) and support the statement with a user-defined assumption. Clearly, the presence of albumin in the urine (observation: _A3.concentration = +) is inconsistent with this assumption and an appropriate diagnostic candidate will be found. So we are left with the diagnosis that part of the glomeruli are broken.

However, if we have additional information about the temporal development, say a positive trend in albumin concentrations, we can employ a different encoding of observations - and have GDE find a reason for the deterioration of the glomeruli. In analogy to the stenosis example above, we can add derivatives and state d/dt _A3.concentration = increasing. While a quantity without an assigned default can take on any value, it cannot change without influences (see the discussion in section 7.2.2.1). Influence resolution for the derivative will generate the assumption CWA(G.broken) that supports d/dt G.broken = const. Prediction progresses in much the same way as for the first scenario and yields CWA(G.broken) as a "single fault" candidate.

This will trigger the search for adequate backward completions and we come up with two behavior constituent types in a properly designed domain theory, namely a version of arteriosclerosis specifically damaging the glomeruli (called nephrosclerosis) and an infection of the glomeruli (glomerulonephritis). See figure 8.35 for a simple representation of the respective behavior constituent types.
Given these model fragments the search for causes of the observed increasing(!) proteinuria will discover either a toxin causing nephrosclerosis - together with high blood pressure within the kidney damaging the glomeruli - or an infection of the kidney that contributed to the deterioration of the delicate tubules filtering the urine.

We have seen how complex medical phenomena can be tackled with the enhanced modeling and reasoning capabilities provided by G*DE. Even feedback loops and degradation faults, which are mostly considered tough problems, are handled gracefully. At least some of the objections put forth in [Patil 1988] should be made obsolete with the presented theory.
8.3 Electrical Circuits Examples

Finally, we will consider two examples from the domain of electrical circuits, which is traditionally the territory of component-based approaches. The classical modeling ontologies are well-suited for most troubleshooting tasks and have proven to be very efficient. However, we will show how the proposed generalization can add valuable new opportunities and solutions.

Our first example deals with a specific kind of interactions that transcend the circuits diagram, namely thermal phenomena between resistors in close proximity. These interactions are "non-local" from the simple component-terminal point of view and, thus, fall within the class of structural faults in the widest sense. Additionally, the example shows how model fragments for phenomena from different physical domains - electricity and thermodynamics, respectively - can be properly combined in a modular fashion.

The second example demonstrates the power of reasoning about configurations. The well-known series-parallel reduction of resistive networks supports prediction by recognizing and exploiting configurations of primitive elements. This procedure can be reconstructed as an application of G*DE, thus giving a prototype of structural reasoning within the presented framework.

Besides giving an idea of modeling techniques for the extended framework, the examples provide a generic domain theory for reasoning about resistive networks and a solid basis for extensions towards a wider class of electrical components. Furthermore, they point at how to integrate electrical models within a more general system context.

8.3.1 Thermal Phenomena in Electrical Circuits

Early after the technique of consistency-based diagnosis was developed, one of the basic assumptions, namely the integrity of the component connection structure, was challenged for the first time ([Preist/Wellham 1990]) - the diagnosis of "structural faults" became an active research topic. However, the inherent limitations of the approaches in the tradition of GDE were not easy to overcome. One of the most promising methods for tackling this kind of problem, [Böttcher 1995], where "hidden interactions" are modeled in strong analogy to behavior constituents, will be discussed in section 9.1.5.

Here, we present a small example including thermal phenomena between two resistors, that establish a "non-local" interaction. See figure 8.36 for a diagram of a circuit with two independent segments. The proximity of the resistors, \( R_1 \) and \( R_2 \), is to suggest that heat can be transferred from one to the other - but this is not given as input. The observations are that one of the light bulbs, \( L_1 \), is much brighter than usual (\( \Delta L_1\text{.brightness} = + \)) while the other light bulb is less bright than in normal operating conditions (\( \Delta L_2\text{.brightness} = - \)). For the enhancement of behavior models by deviations, or \( \Delta \)-models, see section 7.2.5.3.
Diagnosis with a model that relies solely on the conductive connections will certainly end up with a fault in each circuit segment, since at least two disjoint conflicts will be found. Taking thermodynamic effects into account, we can come up with yet another solution: there is a fault in battery $B_1$ generating a much higher voltage than expected (which explains the brightness of $L_1$), this heats up resistor $R_1$ and the neighboring $R_2$. The resistance of $R_2$ will rise with temperature and, thus, reduce current in the lower circuit segment. $L_2$ will be dimmed without a secondary component fault.

We will discuss some issues in modeling electrical circuits - and related thermodynamic phenomena - and show how $G^\ast DE$ can find the mentioned solution when supplied with an appropriate domain theory.

Figure 8.37 depicts a straightforward encoding of the circuit structure of one of the segments from figure 8.36. As shown in section 5.3.3.1, the concept of terminals provides an easy way to express the connection of components. The relations $\text{conn}$ and $\text{conn-from}$ are used to attach terminals to components - and the direction of various elements, like resistors and lamps, is represented by using one tuple of each relation.

Still, there is a number of reasons why we prefer a different encoding of the circuit structure, namely in the form of an SP tree (SP from "series-parallel reduction"). In this representation, series and parallel configurations are explicitly included in the form of a hierarchy. The following section (8.3.2) will both discuss the benefits of this representation and present an approach of how to generate the SP tree from the component-terminal structure within the framework of $G^\ast DE$ behavior constituent types. Figure 8.38 shows the SP tree structure of our example scenario.
There are two aggregate elements, SA1 and SA2, used to encode the serial connection of the respective resistor and lamp. The voltage sources B1 and B2 each supply one of the segments, i.e. the serial aggregate. Note that we have not included any information about spatial proximity and, therefore, the segments are still separate trees.

The full ontology for electrical circuits employed here (in this and the following example) is shown in figure 8.39. The general object type CircuitElement comprises both the type VoltageSource, a connective TernaryNode and ResistiveAggregate. The latter type comprises both primitive elements like Resistor and Lamp and serial or parallel aggregates. Aggregates can be composed of other aggregates (relations part-of and part-of-inv, depending on directions), which covers also the primitive types. It has been mentioned above, that circuit elements can be connected using terminals and the relations conn and conn-from. A voltage source supplies a resistive aggregate - the relation for a connection in the opposite direction is called supplies-inv (for a discussion of directions, please refer to section 8.3.2). Finally, the relation heat-connected provides the basis for heat transfer between two resistors.

Next, the quantities associated with the various object types can be seen in figure 8.40. All resistive aggregates are described by their (overall) resistance, current, and voltage (drop). The primitive element
Resistor has an additional quantity temperature. Voltage sources maintain a voltage and a Lamp has some observable brightness.

![Figure 8.40: Quantity associations for the electrical domain](image)

Now, we will have a look at the behavior constituents governing the basic electrical behavior of a circuit. The following figure 8.41 represents the fundamental law of Ohm.

![Figure 8.41: Behavior constituent type "Ohm's law"](image)

For a lamp, the brightness is assumed to be in a strictly monotonic relationship (constraint type POS) with the current passing through the lamp. See figure 8.42 for the respective behavior constituent type.

![Figure 8.42: Behavior constituent type "Lighting"](image)

Now we come to the thermodynamic effects that can occur when resistors are involved. Figure 8.43 presents two related behavior constituents, namely ohmic heating and the temperature-dependence of the resistance of a (metallic) resistor.
Ohmic heating describes the process of electrical heat generation. This is what is going on in an electrical stove - and by adding the quantity condition that the current has to exceed some threshold (constraint type \texttt{GREATER-x}), we limit the activity to resistors that are under high current loads. This relieves us from modeling standard cooling that keeps the temperature low under normal conditions. In most metallic conductors, the temperature coefficient of resistance, TCR, is positive. This means that with rising temperature, the resistance increases. Again, a threshold condition avoids activation of this behavior constituent for normal conditions.

Heat transfer (figure 8.44) is the central behavior constituent of the thermodynamic interaction between two resistors. The rate of the heat flow is dependent solely on the difference in temperatures. Note that the structural conditions include an explicit heat path (relation \texttt{heat-connected}) between the resistors. For the sake of completeness, we also provide behavior constituents for serial and parallel aggregates, which are the equivalents of Kirchhoff's laws in the SP tree representation. See figures 8.45 and 8.46 for the rather large graphical descriptions.

In figure 8.45, a serial aggregate, SA, with two subaggregates, RA1 and RA2, is shown. Both our manually edited scenarios and the structures resulting from the automated SP tree generation in section 8.3.2 create exclusively binary SP trees, i.e. a given aggregate consists of exactly two parts. However, the flexibility of the influence resolution mechanism allows for the definition of behavior constituents with the quantities of the parts influencing the quantities of the aggregate instead, so that aggregates with an arbitrary number of parts can be used. For the sake of simplicity, we restrict ourselves to the binary case. The laws are easy to understand, if one keeps in mind that the aggregate represents two resistors (that can be aggregate abstractions as well) in series.
Figure 8.45: Behavior constituent type "Serial aggregate"

The analogous behavior constituent for parallel aggregates can be seen in figure 8.46.

Figure 8.46: Behavior constituent type "Parallel aggregate"

The constraint type $\text{FPAR}$ is defined as (a qualitative abstraction of) the following function:

$$\text{FPAR}(x, y) := \frac{x \cdot y}{x + y}$$
so that
\[
\frac{1}{\text{FPAR}(x, y)} = \frac{1}{x} + \frac{1}{y}
\]
which is the relation of the aggregate resistance to the single resistance values for the individual resistors. Even with a qualitative abstraction of resistance values, this expression gives useful predictions in many cases, e. g. when working with deviation models (see section 7.2.5.3).

Finally, the effects of a voltage source "supplying" a resistive aggregate, i. e. the abstraction of the complete circuit, have to be defined (see figure 8.47). In proper connection, the overall voltage of the circuit is simply set equal to the voltage of the source. See section 8.3.2 for a short discussion of sign conventions and the issue of directions.

![Figure 8.47: Behavior constituent type "Voltage source"](image)

Equipped with this arsenal of behavior constituent types, G^DE can attack the scenario with the structure of figure 8.38 and the pathological observations
\[
\Delta L1.\text{brightness} = + \\
\Delta L2.\text{brightness} = -
\]
mentioned above. We will not go into the details of supplying components with some analogy of modes in our framework, but we can instead state normal conditions (e. g. \(\Delta B1.\text{voltage} = 0\)) supplied with a user-defined default assumption (e. g. \(\text{ok}(B1)\)). This is equivalent to considering some component faults - but excluding others, like the battery blocking current altogether. The temperature quantities of the resistors are the only variables that can be influenced by a behavior constituent, so they are the only ones marked as closed (by an explicit mechanism or by use of dummy influences, see section 7.2.2.1) with \(\text{CWA}(R1.\text{temperature})\) and \(\text{CWA}(R2.\text{temperature})\) as the respective closure assumptions.

We will simply present the intermediate result of the diagnosis step. The set of minimal candidates generated by the GDE core looks like this:

\[
\{\text{ok}(B1), \text{ok}(B2)\}, \{\text{ok}(B1), \text{ok}(R2)\}, \{\text{ok}(B1), \text{ok}(L2)\}, \{\text{ok}(B1), \text{CWA}(R2.\text{temperature})\} \\
\{\text{ok}(R1), \text{ok}(B2)\}, \{\text{ok}(R1), \text{ok}(R2)\}, \{\text{ok}(R1), \text{ok}(L2)\}, \{\text{ok}(R1), \text{CWA}(R2.\text{temperature})\} \\
\{\text{ok}(L1), \text{ok}(B2)\}, \{\text{ok}(L1), \text{ok}(R2)\}, \{\text{ok}(L1), \text{ok}(L2)\}, \{\text{ok}(L1), \text{CWA}(R2.\text{temperature})\} \\
\{\text{CWA}(R1.\text{temperature}), \text{ok}(B2)\}, \{\text{CWA}(R1.\text{temperature}), \text{ok}(R2)\}, \{\text{CWA}(R1.\text{temperature}), \text{ok}(L1)\}, \{\text{CWA}(R1.\text{temperature}), \text{CWA}(R2.\text{temperature})\}
\]

Some of these candidates could be refuted by explicit fault models, e. g. \(L1\), as the lamp shining too bright would hardly have a fault mode generating this behavior out of normal current and voltage values.
The most important observation is that the closed-world assumptions for the resistor temperatures show up in some of the candidates. This opens the way for a search for additional influences and if the relation heat-connected is marked introducible, there is a single augmentation of the structure that will refute the closed-world assumption. In figure 8.48, the solution scenario is shown.

![Figure 8.48: Structure of the solution scenario](image)

An extreme interpretation of the scenario even allows for the thermal interaction to be the only cause of the observed deviation: Regard the final intermediate candidate \(\{\text{CWA}(R1\text{.temperature}), \text{CWA}(R2\text{.temperature})\}\) with both assumptions being refuted by the extended scenario in figure 8.48. Imagine that one of the segments \{B2, R2, L2\} is operating at a much lower "normal" temperature, e.g. in a cooled case. If then the thermal connection is accidentally established (e.g. by a break in the casing), the inner resistor, R2, could be heated, while the outer one, R1, could by actually cooled down, so that the respective lamps would exhibit the observed deviations in brightness. Thus, the "structural fault" would account for both conflicts and represent a single fault appearing as a double fault to a traditional diagnosis system, as is a typical case. In a purely qualitative analysis, we cannot easily distinguish between this scenario and the one with \(\Delta B1\text{.voltage} = +\). For that, additional quantitative information about normal temperatures and the respective thresholds is needed.

Admittedly, the example is simplistic in that there is only a single pair of resistors that can be connected by a heat path at all. But like [Böttcher 1995] we would like to leave open the question of how to determine possible interaction paths in more complex scenarios. A delegation to a spatial reasoning subsystem, like proposed by Böttcher, could either be used to choose among proposed additional relation tuples or it could even be integrated with an advanced method to determine the introducibility of a structural element, thus filtering within the search procedure.

Obviously, one can model the thermal interaction of resistors in the component-oriented framework, as well, by simply introducing an additional terminal for the thermal connection. Still, the structure would have to be known a priori and remain fixed - so no unexpected connection of this kind can be dynamically found. Thus, we argue that an extension like the one proposed in this work is both necessary and also more elegant in enabling a modular structure of the domain theory. The thermal effects can be factored out from the electrical ones - the resistor model doesn't have to integrate the temperature-dependency of the resistance with Ohm's law. This becomes even more important when considering interactions between largely different phenomena like hydraulic, pneumatic, thermodynamic, electric, electronic, or even chemical and biological effects, especially if they are interacting closely and cannot be properly separated into different subsystems. See section 8.4 for further examples.
### 8.3.2 Series-Parallel Reduction in Electrical Circuits

The previous example relies on an SP tree representation of the circuit structure, which has several advantages over a connection-based one. See [Mauss/Neumann 1996] for a detailed discussion of the benefits, with probably the most prominent one being the ease of (complete) prediction by local constraint propagation. As a matter of fact, most diagnostic reasoning systems have to cope with incomplete predictors as already mentioned in [de Kleer/Williams 1987]. Local constraint propagation, as employed in GDE, and in a modified form also in the G+DE prototype, proves to be very efficient, especially in integrating assumption tracking, but has some deficits in making all possible predictions in certain complex structures.

This can be compensated by structural enhancements. The key idea is to recognize and exploit certain configurations in the given structure and add additional constraints, which do not represent any new effects, but rather support reasoning about the existing structure. It turns out that the modeling paradigm of behavior constituents is well suited for this kind of enhancement. Note that this is an example for a novel application of the G+DE framework rather than of a particular system model to be diagnosed.

The original technique of generating SP trees is described as a "reduction" of a given circuit by replacing resistive elements in series or parallel by a single resistor. This replacement resistor can be further aggregated, until finally a single resistor represents the original circuit. The subsumed resistors are stored in a hierarchical structure, the SP tree, so that predictions made for the aggregated elements can then be propagated back down to the original structure.

As a precondition for the procedure to succeed, there is to be only a single source and the network has to be "SP reducible". A circuit that is not SP reducible necessarily contains a substructure with all nodes having a degree of 3 and above. For a precise definition, please refer to [Mauss/Neumann 1996], where also the star-mesh conversion is discussed, as a means to overcome this limitation. For our example, we will stick with the simpler case.

In figure 8.49, an overview of the three basic enhancements is shown in an informal notation. As input, we assume a structure with binary terminals, i.e. there are at most two elements connected to each terminal. Branches or "stars" are to be modeled with one or more ternary nodes. The diagram ignores directions and sign conventions, see below for a discussion of these issues.

![Figure 8.49: Overview of the aggregation enhancements (informal notation)](image)

To the left of figure 8.49, the aggregation of two elements in series is described. The elements can be aggregates themselves (object type ResistiveAggregate). The dotted lines denote the elements to be introduced, namely the replacement aggregate, which is connected to the outer terminals itself (so it matches the structural conditions of further aggregations), and the part-of relations with the original elements, denoted as dotted lines with a diamond at the aggregate. In the middle, the parallel aggregation can be seen. Again, a single aggregate element is introduced, connected to the outer terminals - including the ternary nodes - and the part-of relations are established. Finally, the right part of the diagram defines...
that a single aggregate connected to a single (voltage) source is in the supplies relationship with the source.

Next, the formal definitions of the respective behavior constituents are presented. Figure 8.50 contains the serial aggregation. Note the directionality expressed by the distinction of the conn and the conn-from relation tuples - it reflects also in the serial aggregate and the newly established connections to the outer terminals.

![Figure 8.50: Behavior constituent type "Serial Aggregation"](image)

As resistors and resistive aggregates can be connected in any direction, a second very similar behavior constituent (see figure 8.51) is needed, where the second original aggregate, RA2, is connected in reverse (conn to terminal T2, conn-from terminal T3) - and to recognize the difference, a different version of the part-of relation is introduced between RA2 and SA12, called part-of-inv. This will have an impact on the signs of resistances, currents and voltages for the behavior constituent type "Serial Aggregate" (cf. figure 8.45). This is not spelled out here, as it adds very little to the understanding of the overall process.

![Figure 8.51: Behavior constituent type "Serial Aggregation Opposite Directions"](image)

The following behavior constituent type "Parallel Aggregation" (figure 8.52) introduces the parallel aggregate and the part-of relationships, as well the connection to the outer terminals. The latter effect is needed only for the new aggregate to meet the structural conditions of further aggregation behavior constituents - it does not imply that there is an additional connection: the terminal is still understood as a binary one, and all propagation of values through the structure is done along the SP tree, i. e. through the behavior constituents shown in the previous example (figures 8.45, 8.46, and 8.47) mediated by the relation part-of (and part-of-inv).
This one also has a second version with the second sub-aggregate (RA2) reversed (see figure 8.53). In this way, the first sub-aggregate determines the direction for the new parallel aggregate (as it does for the serial one, see above) - and ensures that only a single behavior constituent type matches any given pair of parallel elements. (But see the discussion below for multiple overlapping SP trees).

Indeed, with the final behavior constituent type that will determine the supply of the topmost aggregate of a circuit section by a single(!) voltage source (see figure 8.54 for both direction versions), the set is almost complete. We have deliberately left out behavior constituents dealing with the special cases of an "end branch", that would correspond to a terminal connected only on one side, and "self loop", a terminal connected to the same aggregate on both sides. See [Mauss/Neumann 1996] for the treatment of these cases.
The presented set of behavior constituents will enhance any SP reducible structure given with the complete SP tree, so that the model fragments of the previous example can be used. However, it is important to mention that there are often multiple possible SP trees and the original "SP reduction" procedure contains a non-deterministic element in choosing one of them. However, the defined semantics of G^DE provide no such facilities for choice and non-determinism. In effect, all possible SP trees are constructed - which might be very inefficient in case of a high degree of overlap, but is provably correct. Remember that all additional constraints introduced by the aggregates are only rewrites of the original ones. So each consistent value assignment for the original network is guaranteed to also satisfy all possible SP trees.

See the following diagram (figure 8.55) for an example of how a slightly larger circuit is enhanced by creating a set of aggregates. In this case, the SP tree is unique - so the behavior constituents do not produce superfluous overlapping aggregations. We use the compact and less formal notation that also neglects direction and sign conventions. The aggregates are created in the order SA23, PA234, SA1234 and finally the supplies relation with the source is established.

To emphasize the usefulness of the presented enhancement, [Mauss 1998] shows not only how the integration of the star-mesh conversion overcomes the limitation of "SP reducibility", but also how a
generalized concept of aggregating characteristic lines extends the approach to multiple sources, as well as other types of components beyond resistive elements.

See the previous example (section 8.3.1) for how the original terminal-connections structure can be completely neglected when instantiating behavior constituents. No constraints or other quantity effects are created for a terminal and its neighboring components. Although this would be possible and would constitute a more traditional way of using "component models" (see figure 5.10 for an example), in our case the model composer creates the SP tree first, and then establishes the constraint network exclusively from the aggregation hierarchy. We have shortly mentioned the advantages of this procedure, especially when employing predictors based on local propagation - and the richer modeling paradigm and composition capabilities presented in this work provide elegant means for exploiting this.
8.4 Applicability in Further Domains

We have presented a small set of examples from different domains and with different focus. This should give an idea of how the applicability of consistency-based diagnosis is extended to novel domains by the generalized modeling paradigm and the more powerful reasoning machinery proposed here. It is not by accident that many of the ideas leading to the development of this theory and ultimately the G^DE prototype have been triggered by problems in hydro-ecology. In several collaborative projects, we have seen the appeal of the formal rigor of consistency-based diagnosis and felt the urge to employ the efficient implementations existing - but the modeling problems posed by the complex interactions of physical, chemical, and biological phenomena, distributed in space and time presented a serious obstacle. Component-oriented models were simply inadequate for all but the simplest scenarios. However, rather than abolishing the use of consistency-based techniques, we have developed an extension that proved helpful and successful in a much wider sense. The theory of extended consistency-based diagnosis, the flexibility of process-oriented models and the reasoning power of G^DE, yet to be exploited to full advantage, are applicable in a variety of new and old domains.

Ecological reasoning in the strict sense, i. e. theory building about the interactions within the ecosystem, could benefit from the facilities to check hypothetical models for consistency and, maybe even more important, for completeness. In many cases, it is contradictions of predictions to the observations that drives ecological research, resembling diagnostic reasoning on a larger scale. Process-oriented modeling is also well-suited for the domain of thermodynamics - the Qualitative Process Theory ([Forbus 1984]) was developed largely with this problem field in mind. See [Collins/Forbus 1989] for an insightful and comprehensive modeling achievement, which also serves as a good tutorial in modeling. The only reason why the interesting models of phase transition systems such as refrigerators and thermal control systems have not yet been carried over to the G^DE framework is that they rely heavily on global comparisons and corresponding values.

The ease of integration of model fragments from different disciplines in our theory is to be emphasized. Chemical engineering typically involves tasks, where this is necessary. And, indeed, a collaborative project initiated by a chemical engineering department of a renowned Indian research institute has been proposed, where generalized diagnosis techniques are to be employed for tasks such as systems analysis, model revision and technical diagnosis.

Several European car manufacturers and associated systems suppliers have participated in national and international research projects with a diagnosis focus. Automotive technology has become a complex field with tight coupling of electric, pneumatic, and hydraulic systems - with more and more electronic control. The author has participated in one of these projects for three years and gown increasingly aware of the problems created by considering each subsystem in separation - while many faults can be only understood from an integrative perspective. This is to be a necessary next step in automotive diagnosis and maybe the theory presented here can help supporting this integration with computational means.

Finally, many so-called "soft domains" such as economics, sociology or psychology have not been susceptible to logical theories of diagnosis - even so they could profit from computational support to theory building and model revision. Hopefully, the proposed extension of the theory facilitates its adoption in areas where models are in extensive use, but model building and revision traditionally remained a tedious manual task.
9 Conclusions

In this final section, we present a short survey of related work (section 9.1), before summarizing and discussing the presented approach (section 9.2). Finally, perspectives for future work and possible extensions to theory and implementation are mentioned (section 9.3).

9.1 Related Work

First, we will discuss some important approaches to model-based diagnosis from the perspective of the one presented here. This serves the dual purpose of paying tribute to the various ideas that have been influential for our work and of stressing the generality of the extended set of techniques presented here by relating them to existing ones.

9.1.1 GDE

First and foremost there is the groundbreaking work of Reiter ([Reiter 1987]) and others ([de Kleer et al. 1992]) who did not only lay the foundations of the field of model-based diagnosis, but also implemented the General Diagnostic Engine, GDE, ([de Kleer/Williams 1987]). All later attempts to logically define the diagnosis task, including ours, invariably recur to the basics laid out in the cited research papers, where we first encounter the typical consistency-based characterization of the diagnosis problem as

\[ SD \cup OBS \models \bot. \]

We already have discussed the strong assumptions and limitations inherent to the GDE approach in section 3. The central issue of being focused on components (and their failure) as a built-in ontology can now be seen as a specialization of the presented concept of behavior constituents. As a matter of fact, a GDE-style algorithm forms the core of the GDE implementation (see section 7.3), which is evidence that the presented approach is strictly more general in its applicability. Thus, every system and every fault of that system that can be diagnosed with GDE can also be diagnosed with G+DE - and the models are quite straightforward to adapt.

Section 5.2 discusses this issue and the examples in section 8.3 show how G+DE components can use and transcend component-oriented models. By reducing behavior constituents to simple component models again, the structural conditions become very simple (see section 5.3.3 for how to encode the useful concept of "terminals"), while quantity conditions are completely absent. The quantity effects are restricted to constraints and structural effects are excluded altogether.

Regarding the effort spent in G+DE to compose, resolve and transform the behavior model, we find that GDE neglects these tasks, since they become trivial. The occurrence rules stated in section 5.3.3 collapse to

\[
\begin{align*}
(OR_1') & \quad SC(bct, m) \rightarrow \exists bc \text{ BehaviorConstituent}(bc, bct, m) \\
(OR_2') & \quad \text{BehaviorConstituent}(bc, bct, m) \rightarrow QE(bc)
\end{align*}
\]
which makes the complex transformation described in section 7.1 superfluous, or from another perspective, the extended forward space corresponds exactly to the forward completion. But the simplification becomes even clearer when looking at the backward completion - which is not an issue for GDE! There are no structural augmentations to be discovered - and no closed-world assumptions to detect them, since influence resolution is omitted (QE does not contain influences!). Faults are strictly diagnosed in terms of "user-defined" assumptions. The most obvious way to attach ok assumptions to model fragments is by including antecedents in the constraints in QE, much as has to be done for the generic first-order sentences described by [Reiter 1987]. Thus, the G⁺DE architecture essentially collapses to the GDE core, which is not surprising at all.

### 9.1.2 GDE⁺ and Sherlock

GDE⁺ ([Struss/Dressler 1989]), and Sherlock ([de Kleer/Williams 1989], also see section 3.2) both overcome an important limitation of GDE by introducing fault models, so that diagnostic candidates can be checked for consistency with the observations. This leads to significantly better results, especially if strong models of faulty behavior are available - as impossible candidates can be ruled out and the remaining ones usually give a better picture of what is wrong with the system at hand. In addition to fault detection and fault localization, fault identification becomes possible. As a tradeoff, higher computational complexity and increased modeling effort have to be taken into account.

While GDE⁺ and Sherlock excel in clever instantiation of fault models for only a small set of components under strong suspicion and exploit a priori probability information for focusing, in principle the power of fault models can easily be achieved in the G⁺DE paradigm: Either a component's mode becomes a simple variable within a behavior constituent encoding the complete set of modes, or a component type in different modes is represented by different object types, thus triggering different behavior constituents (corresponding exactly to the primitive ok models or fault models). In the latter case, one has to take care that the "replacement" of a component by one of its faulty versions is interpreted (and weighted) as an elementary step - rather than counting it as a missing and an unsuspected component at once.

Our extended theory can be restricted to the GDE⁺ case again. The form of the occurrence rules is depending on the way component modes are encoded. When using different object types, we end up with the same formulas as in the GDE case - the difference is to be seen solely in the revision module. However, when modes are represented by quantities, we have to include quantity conditions and get

\[
\text{(OR}^\dagger\text{) } \quad \text{SC}(bct, m) \land \text{QC}(bct, m) \rightarrow \exists \text{BehaviorConstituent}(bc, bct, m)
\]

\[
\text{(OR}^\dagger\text{) } \quad \text{BehaviorConstituent}(bc, bct, m) \rightarrow \text{QE}(bc)
\]

Note that the simple form of QC (and the fact that they are defaulted to ok) encourages to take them into account in model composition. In this way, the extended forward space could represent the forward completion under a given mode assignment. Control strategies could be adapted to emulate original GDE⁺ behavior, although such a hybrid is advisable exclusively, if the additional power of G⁺DE is necessary for the task at hand, e. g. to account for faults by additional elements or unexpected interactions (compare the example on thermal interactions within electrical circuits in section 8.3.1).

A probability (or preference) rating could be helpful for the process-oriented approach, as well. Closed-world assumptions usually would be the least preferred elements within diagnostic candidates, thus preferring all solutions within the forward completion - up to a limit where a small backward completion is a more plausible solution. See the diagnosis of structural faults in section 8.3.2, where multiple apparent component faults are discovered to be caused by a single bridge fault.
A closed set of fault models is a special case of a closed domain theory - at least if it does not contain the ominous "unknown fault" discussed in both papers cited above. Adding a fault mode that does not impose any constraints on the components (terminal) variables, i.e. has no logical consequences, amounts to opening the domain theory to allow for a form of "suspension" as a last resort. Assigning this fault mode (usually given a very low probability) can be seen as opening a closed-world assumption - without assuming that anything can be said about its negation. One should be aware that the closure of the domain theory potentially places even stronger requirements on the modeler than providing a complete set of fault models, since all relevant phenomena within the domain have to be accounted for.

9.1.3 DDE

The Default-based Diagnosis Engine, DDE, ([Dressler/Struss 1994], also discussed in section 3.2) further extends the abilities to exploit additional information about the possible faults of components. Modes can be structured as a hierarchy according to a preference relation. A mode is to be considered if and only if all modes strictly preferred over it have been ruled out. This yields powerful control over the diagnostic process and avoids unnecessary instantiation of faults. The semantics of the approach have also been formalized in Default Logic as a set of simple preference defaults - and the implementation relies on a non-monotonic extension of the ATMS ([Dressler 1990]) to manage these dependencies. Again, the emphasis of the work on DDE, and especially of the paper [Dressler/Struss 1994] lies on effective control strategies that avoid instantiation of fault models (by focusing candidate generation and testing). This is achieved by exploiting the restriction that devices are modeled as fixed component structures - and the assumption that component faults occur mostly independently. In the presented theory of process-oriented consistency-based diagnosis, these focusing principles cannot be simply adapted, but we admit the necessity of further work in this direction (see section 9.2 and the perspectives).

9.1.4 PDE

The work of Collins ([Collins 1993]) is the one that comes closest to the presented approach. Models built within the original Qualitative Process Theory ([Forbus 1984]) are used as input to the Process-based Diagnostic Engine, PDE, which employs a qualitative envisioning component for prediction and a modified ATMS with a rule-based interface for the generation of diagnostic candidates. Collins employs local closed-world assumptions in much the same way as we do - and they will be collected in intermediate diagnostic candidates, as well. Model closure is formulated as explicit "predicate closing", an alternative non-monotonic logic framework, and implemented as the generation of ATMS justifications for the closure formulae. A closed-world assumption is added and - as happens with all other assumptions - recorded in each conclusion by ATMS label propagation. A hitting set algorithm calculates the candidates from discovered conflicts, which is largely equivalent to the standard GDE core component of G+DE.

However, once diagnostic candidates containing closed-world assumptions are discovered, Collins proceeds in two steps by first hypothesizing a "process structure", i.e. a set of (active) process instances that could account for the opening of the closure - and then searching abductively for an underlying object structure to imply the desired process structure. He has devised his own ATMS-based abduction algorithm, which seems to be well suited for best-first focusing - and importantly, "new" structural elements ("skolemized" in Collins' terminology) are given a low priority, so that existing ones are always
preferred. In this respect, PDE is more advanced, as it employs a more focused search strategy taking quantity values into account.

In its presented form, PDE relies on a single snapshot in the behavioral development of the system under consideration, just as we do. Observations and predictions all refer to the same time frame, which makes it easier for the consistency check. It seems that neither approach features an advantage for future inclusion of temporal aspects.

Apparently, PDE is not built for handling general structural effects - or "contingent objects" in QPT terminology. Even in QPT the handling of these effects remains unclear. We have pointed out the inconsistencies in introducing contingent objects in QPT views and processes (see section 5.1 and "logical existence" versus "real existence"). An analysis of the "incremental qualitative envisioner" ([DeCoste/Collins 1991]) used by PDE shows that changes in the structure are left to external control, in the form of assumption sets. But Collins neither discusses the complex issues involved, nor are they featured in any of his examples. A review of sections 6 and 7 reveals the extent of the simplification thus achieved.

Generally, the diagnostic approach of PDE is diligently formalized, but often in intricate connection with the justification structure of the ATMS. The presentation of the modeling approach is less formal and heavily relies on LISP notation. Unfortunately, all application examples are quite simplistic, mostly concerning the detection of leaks and clogging in simple multi-tank structures. Possibly, this has contributed to the regrettable effect that work towards the combination of process-oriented modeling and consistency-based diagnosis have remained largely unknown in the research field.

9.1.5 Diagnosis of Structural Faults

There have been various attempts to overcome the dependency of component-oriented diagnosis on a fixed structure and to cover "structural faults". A typical approach is to represent all potential structural faults, e.g. bridges between wires (or "bridge faults" in integrated circuits, which behave differently) as explicit components, basically of an insulator type, that can fail in establishing a bridge (or an AND-gate in TTL technology). [Preist/Wellham 1990] is representative for this technique, and they also provide a discussion of the benefits of the additional effort of providing a component instance for each location where bridges can occur. Their basic argument is based on the frequency of such structural faults in relation to simple component failures.

[Böttcher 1995] instead represents a potential interaction type as a special kind of model fragment, a "hidden interaction", which has conditions in terms of structural configurations (the representation and checking of these is left to a separate domain-specific module) plus conditions on the modes of the components involved. Its effects are given as constraints. The cited paper revolves around the clever integration into an advanced consistency-based diagnosis framework in the form of "working hypotheses" (formalized in [Struss 1992b] and refined in [Böttcher/Dressler 1994]), so that the instantiation of the interaction model fragment can be delayed until it forms a preferred hypothesis - and this can be done generically for every matching configuration.

The definition of "hidden interaction models" is surprisingly close to what we would call a behavior constituent type definition. In [Böttcher 1995] we find the abstract defining formula

\[
\text{hidden-interaction-hypothesis}(c_1, \ldots, c_n) \rightarrow \\
\text{constellation}(c_1, \ldots, c_n) \land \\
\text{mode-constellation}(c_1, \ldots, c_n) \land \\
\text{connection}(c_1, \ldots, c_n)
\]
The constellation being a spatial or geometrical description of the component layout, not being specified within the framework, but rather representing an interface to a specialized reasoning module. The modes compatible with the hidden interaction are defined in the predicate `mode-constellation` and, finally, the connection describes the constraints of the interaction behavior.

A reconstruction within our framework is simple, but depends on the chosen encoding of fault modes (see section 9.1.2 above). Either they are referenced in the structural conditions (as object types) or the quantity conditions. Our explicit ontology also provides the means to directly specify the structural configuration (`constellation`) of the components using relations. Note that the absence of a mechanism for the composition of partial behavior specifications, as provided by influences in our approach, poses a problem for the specification of the interaction effects. The constraints defined in `connection(c_1, ..., c_n)` have to match the respective fault models of the individual components, i.e., if a leak is described as a hidden interaction, also leaking models for the participating components have to be provided.

Nevertheless, in this approach faults can take on a more complex form - involving multiple components in a certain configuration and state in order to occur and result in "non-local" effects, at least from the perspective of standard terminal connection structures. However, the correct behavior is still restricted to being modeled as a one-to-one mapping of components to model fragments. Clearly, the process-oriented modeling scheme presented in this thesis goes a step further. Leaving aside an analysis of the efficiency of such a model in G*DE, it should be obvious that our approach is more powerful, e.g. in discovering such interactions from their observed effects. See section 8.3.1 for an example of diagnosing structural faults.

### 9.2 Discussion

In this section, we summarize the main limitations (section 9.2.1) and advantages (section 9.2.2) of the presented theory and implementation.

#### 9.2.1 Limitations of the Approach

One of the main limitations of the presented theory is the state-based approach: all structural statements as well as all observations and predictions refer to the same instant in time, which largely facilitates consistency checking. Explicit representation of time and change would require a simulator as a predictor, but it would also enable to model real creation and destruction of elements, as well as the ability to generate temporal therapy plans, in order to move the system through a sequence of states towards a desired one.

Note that the enhancement of a model with derivatives or deviation models partly compensates the need for temporal representations. See also the fundamental results of [Malik/Struss 1996] and [Struss 1997] for the diagnosis of dynamic systems without with state-based approaches. The latter paper derives formal conditions for the equivalence of state completion and simulation in a diagnostic context.

A concern arising with the current prototype is the impact of the model composition on efficiency. For complex domain theories, potentially a large number of behavior constituents are collected, while many of them might never be activated. As an example, see the naive reconstruction of component fault modes as quantities (section 9.1.2), which would include every fault model into the initial behavior model - and leave the complete diagnosis task to the predictor. For certain classes of domain theories, of course the
composition step could be extended to take part of the quantity conditions into account, so that the creation of certainly inactive instances would be avoided.

The strong requirements of assumption tracking usually lead to the employment of incomplete predictors in GDE-style implementations. While this problem is not unique to process-oriented diagnosis, it has to be mentioned that missing conflicts can lead to a weaker situation assessment, which reduces the usefulness of the intermediate results as a basis for a therapy recognition step. On the other hand, the powerful model composition algorithms can also be applied to compensate for the incompleteness of prediction by constraint propagation. See section 8.3.2 for an example.

The process-oriented modeling paradigm is certainly a very general one. However, there are certain specifics and restrictions of the modeling language as presented in section 5. We argue that most of them are not inherently limiting for the user. For instance, the fact that quantities can only associated with objects can almost always be overcome by creating an object to carry the quantity, a technique called "reification". The constraint types and influence types allowed are only limited by the ability of the employed predictor to handle them (for influences this also includes the resolver). Our prototype is, thanks to the RAZ'R Runtime System, open for defining completely new constraint types - at least for finite domains.

9.2.2 Advantages of the Approach

It is worth mentioning that our work builds on the rigorous formal foundations of consistency-based diagnosis and extends and generalizes them, rather than adding heuristics or ad-hoc solutions. The concise characterization of the diagnosis tasks (section 6) is evidence for this, which also achieves a necessary differentiation between situation assessment and therapy recognition.

Even more so, this thesis is more specific with respect to the structure of system models allowed (section 5). It thus requires a certain stringency in structure-to-behavior reasoning. In effect, we preserve and enforce the approach from "first principles" the research field is based on.

The central advantage of the presented theory and implementation of process-oriented consistency-based diagnosis is the generality of the approach. It provides a superset of the modeling constructs available for component-oriented diagnosis and reconstructs a major part of the Qualitative Process Theory, including a complete and conclusive treatment of structural effects. This paradigm achieves a more intuitive modeling approach in many domains, while it enables diagnosing others in the first place.

Also, the closure of the domain theory is arguably a more elegant way of excluding arbitrary effects than fault models or "physical impossibility", as proposed in related work (see section 9.1). In describing every possible phenomenon in the class of systems under consideration, it facilitates the search for unexpected elements or interactions in a unique way.

Note that the closure of the system model is a necessary requirement for most prediction and diagnosis tasks. Variable values are always determined by what is known to have an effect on them - excluding everything else. However, usually the system boundary is implicit and will never be challenged. This is one of the areas, where our generalized theory provides a clear and formal alternative in making the closure assumption explicit - as well as its local effects. This is especially important for generating therapy proposals, as human interactions are usually unaccounted for in the initial model.

The system architecture of the G\textsuperscript{DE} prototype (section 7.3) stands for a flexible implementation, that can be adapted to provide diagnosis tools for specific purposes. Model transformations can be added in different stages of the composition process and specific solutions can be configured, e. g. by employing a
simpler composition module, tailored exclusively for the diagnosis of structural faults in component structures.

As evidence of the applicability of the presented theory, as well as the implemented prototype, for a variety of diagnosis tasks and in largely different domains, see the extended examples in section 8. There, also the benefits of the generalized approach for ecology, thermodynamics, chemical engineering, and automotive systems are outlined (section 8.4).

9.3 Perspectives

The prototypical implementation of $G^*$DE can be extended in various directions. To increase usability, a more sophisticated user interface would be helpful, especially for providing graphical feedback on predictions and diagnosis results. This requires automated layout of graphs, as extensions of the user-defined scenario can occur in model composition (forward completion), as well as situation assessment and therapy recognition (backward completion). A fully interactive scenario editor could facilitate the selection, testing and refinement of situation hypotheses and proposed therapies.

Explanation generation by interpreting the justification structure generated in the ATMS is possible. The user could be informed about the reasoning steps that led to the various conflicts and the predictions made on the path. A simple mapping from quantity assignments to natural language expressions forms a promising start, e.g.

\[
\text{IronRedissolving1.active} = \text{true} \\
\Rightarrow \\
\frac{d}{dt} \text{DissolvedIron1.concentration} = +
\]

becomes

"the iron redissolving process from the sediment to the hypolimnion is active implies the concentration of dissolved iron in the hypolimnion increases"

Of course, ultimately answering user queries is a much more sophisticated task. This is a partial answer to the call for "articulate software" by Forbus (e.g. [Forbus/Whalley 1994]), which is useful even beyond educational software and tutoring systems.

The search algorithm for backward completions (section 7.2) can be improved, especially when considering specialized domain theories and system structures. The general case is quite hard to control, since the modeling language is open for a wide range of possibilities. But for particular classes of systems it is often possible to derive useful heuristics and focusing strategies. Also, it might be worth the effort to develop focusing principles for model composition in restricted application contexts. The instantiation of behavior constituents never to be activated can be avoided in many cases, e.g. by considering part of the quantity conditions.

By relying on predefined parts of the domain theory or even the situation description, one can envision problem solvers of varying generality from simple monitoring tools (enter a few measurements, get predictions and/or alarms) via hypothesis testing ("what if there was X in the system?") to sophisticated theory building support (checking new domain theories against a set of scenarios). Certainly, not every user needs to understand the intricacies of locating spatial objects using unique relations.
It is an open issue, how probabilities can be exploited in focusing the reasoning process. Rating candidates containing closed-world assumptions as less likely than the ones consisting solely of user-defined assumptions makes sense in most of the applications. However, discarding candidates or intermediate search results is tricky, since it might be the case that a simple (and likely!) deeper cause is responsible for the strange symptoms.

The current implementation of the "revisables" control mechanism, i.e., by labeling object types and relations as introducible, can be generalized to more complex rules to determine the introducibility of structural configurations. However, the matching algorithm (see section 7.2.4.2) has to be extended to cope with this, probably by look-ahead techniques for checking whether a partial match can be completed in an allowable way.

A controversial idea is to allow the instantiation process to mark participating objects - and to use this in the structural conditions of other constituents. This would allow for non-deterministic composition, which provides advantages for very special applications, e.g., the construction of SP trees, see section 8.3.2. However, the loss of formal rigor (the logical semantics would have to be changed - this corresponds roughly to negative structural conditions) is probably not desirable.

It is an issue of active research, how to integrate temporal aspects or even general simulation and reasoning about action and change into the theory of process-oriented consistency-based diagnosis. One problem posed for temporal reasoning is the discontinuity inherent in the semantics of behavior constituents, as they "switch" on and off in time, as critical quantity conditions become true or false. The other main concern is the semantics of the closed-world assumption, which closes "what is there" - at a particular point in time. Changes in time require a different kind of closure. It turns out, that most existing approaches to temporal diagnosis in the strict sense (for an overview see [Console/Theseider Dupré 1998] or [Brusoni et al. 1998]) do not generalize well to encompass process models, they mostly rely on the structure of the system to remain fixed over time. However, "Multiple Context Temporal Constraint Propagation" (MCTCP, see [Dressler/Freitag 1994]), a generic ATMS-based technique to share prediction results across time and logical contexts, could be applicable.

Finally, a true generalized theory of diagnosis is to describe and formalize the integrated interactive process of situation assessment and therapy recognition. This includes measurement proposal and testing as diagnostic actions, and all of these can occur in any order and have to be planned accordingly. Real-world problem solving usually involves observations, reasoning, changes to the system and new observations and reasoning in an interlocked sequence. See [Struss 1992b] for an early treatment of "diagnosis as a process" and [Malik 2001] for a recent approach to the integration of model-based diagnosis, testing, and repair.
Appendix A: Formal Proofs

In this appendix, we present formal proof for theorems 1, 2 and 2b from section 7.1. They constitute an important part of the evidence that the algorithmic approach chosen in the specification of the Generalized Diagnosis Engine, G^DE, is equivalent to the semantics stated in sections 5 and 6.

A requirement that has been mentioned before (section 7.1.2.2) is a certain stringency of the domain theory, namely that for each behavior constituent type, the (bindings of) structural effects can be uniquely determined from the structural conditions. Usually, this is done by making use of relations with "uniqueness" or "functionality" properties. But instead of describing the restriction on the level of relation properties, the following unification rules give an abstract formal specification. The first rule states the uniqueness of a behavior constituent instance w. r. t. given structural conditions - and the same for potential behavior constituents.

The second rule is different in that it also describes an abstract requirement for the creation of the extended forward space, i. e. the algorithm identifying potential structural effects - relying on the very same stringency in modeling and the use of relation properties - namely that objects from the structural effects are unified with existing objects exactly when they have to. As a remark, this also excludes reasoning about "reclassification" by assuming a different, i. e. more specific type for an existing object. As a rule this is written using the structural equivalence relation, \( \text{Equiv} \), which will be defined below:

**Unification rules:**

1. \( \text{BehaviorConstituent}(bc_1, bct, m_1) \land \text{BehaviorConstituent}(bc_2, bct, m_2) \land \text{Extends}(m_1, m_2, bct) \rightarrow m_1 = m_2 \land bc_1 = bc_2 \)

2. \( \text{PotentialBehaviorConstituent}(pbc_1, bct, m_1) \land \text{PotentialBehaviorConstituent}(pbc_2, bct, m_2) \land \text{Extends}'(m_1, m_2, bct) \rightarrow m_1 = m_2 \land pbc_1 = pbc_2 \)

3. \( \text{SEObj} (o_1, bc_1, or_1) \land \text{SEObj}'(po_1, pbc_1, or_1) \land \text{Object}(o_2) \land \text{PotentialObject}(po_2) \land \text{Equiv}(bc_1, pbc_1) \land \text{Equiv}(o_2, po_2) \rightarrow (o_1 = o_2 \leftrightarrow po_1 = po_2) \)

The predicate \( \text{Extends}(m_1, m_2, bct) \) expresses the identity of the mappings, \( m_1 \) and \( m_2 \), with respect to the object roles from the structural effects of the specified behavior constituent type, \( bct \), see section 5.3.3.1 for the definition. The first unification rule (UR1) states that behavior constituent instances are identical if they have the same structural conditions binding - of course, it would suffice to state identity of mappings \( (m_1 = m_2) \) or of instances \( (bc_1 = bc_2) \), since they imply each other. The analogy to the potential case is obvious.

The second unification rule (UR2) states that the identification of a structural effect object with another object takes place exactly when required by the original semantics (the level of \( o_1 \) and \( o_2 \)). This is mediated by the \( \text{Equiv} \) relation to the level of \( po_1 \) and \( po_2 \), as used in the construction of the structural equivalence below. The model composition algorithm of G^DE (see section 7.2.1) is designed to guarantee this.
We repeat theorem 1 before presenting a proof:

**Theorem 1:**

If there is no backward completion, there exists a subset of the extended forward space that is structurally equivalent to the forward completion of the situation description. Formally:

\[ \forall e \neg \text{IntroducedElement}(e) \rightarrow \]

\[ (C) \quad \text{BehaviorConstituent}(bc, bct, m) \rightarrow \exists pbc \exists m' \text{ PotentialBehaviorConstituent}(pbc, bct, m') \land \\
\quad \text{Equiv}(bc, pbc) \land \\
\quad \text{Element}(e) \rightarrow \exists pe \text{ PotentialElement}(pe) \land \text{Equiv}(e, pe) \]

\[ (M) \quad \text{Equiv}(x, y_1) \land \text{Equiv}(x, y_2) \rightarrow y_1 = y_2 \]

\[ (I) \quad \text{Equiv}(x_1, y) \land \text{Equiv}(x_2, y) \rightarrow x_1 = x_2 \]

\[ (SE_1) \quad \text{RelationTuple}(rt) \land \text{PotentialRelationTuple}(prt) \rightarrow \\
\quad (\text{Equiv}(rt, prt) \leftrightarrow \exists rel \exists o_1 ... \exists o_n \exists po_1 ... \exists po_n \text{ RelationTuple}_n(rt, rel, o_1, ..., o_n) \land \\
\quad \text{PotentialRelationTuple}_n(prt, rel, po_1, ..., po_n) \land \\
\quad \text{Equiv}(o_1, po_1) \land ... \land \text{Equiv}(o_n, po_n)) \]

\[ (SE_2) \quad \text{BehaviorConstituent}(bc, bct_1, m_1) \land \text{PotentialBehaviorConstituent}(pbc, bct_2, m_2) \rightarrow \\
\quad (\text{Equiv}(bc, pbc) \leftrightarrow bct_1 = bct_2 \land \\
\quad (\forall or \text{ Maps}(m_1, or, o) \land \text{Maps}'(m_2, or, po) \rightarrow \text{Equiv}(o, po))) \]

**Proof:**

The definition of the extended forward space via the instantiation rules (IR1) and (IR2) has been described in section 7.1.2.2, with a "copy" of the initial situation description being the starting point. By construction, we can guarantee that for the initially specified set of structural elements, there exists a relation \( \text{Equiv} \), which obeys (M), (I), and (SE1):

\[ (E0) \quad \text{Equiv}(e_1, pe_1) \land \text{Equiv}(e_2, pe_2) \land \text{KnownElement}(e_1) \land \text{KnownElement}(e_2) \rightarrow \\
\quad (e_1 = e_2 \leftrightarrow pe_1 = pe_2) \land \\
\quad \text{Equiv}(rt, prt) \land \text{KnownRelationTuple}(rt) \rightarrow \\
\quad (\text{RelationTuple}_n(rt, rel, o_1, ..., o_n) \land \text{PotentialRelationTuple}_n(prt, rel, po_1, ..., po_n) \rightarrow \\
\quad rel_1 = rel_2 \land n = 1 \land \text{Equiv}(o_1, po_1) \land ... \land \text{Equiv}(o_n, po_n)) \]

We will show how to extend this to the complete forward extension, thus proving theorem 1. For behavior constituent instances, we define an actual one to be equivalent to a potential one of the same type, if and only if all objects from the structural conditions are equivalent:

\[ (E1) \quad \text{BehaviorConstituent}(bc, bct, m) \land \text{PotentialBehaviorConstituent}(pbc, bct, m') \rightarrow \\
\quad (\text{Equiv}(bc, pbc) \leftrightarrow (bct = bct' \land \\
\quad (\forall or \text{ SCObj}(o, bc, or) \land \text{SCOobj}'(po, pbc, or) \rightarrow \text{Equiv}(o, po)))) \]

And for structural elements that are not included in the initial situation description, we define equivalence by creation from equivalent behavior constituents - with identical structural effect roles:

\[ (E2) \quad \text{Object}(o) \land \neg \text{KnownObject}(o) \land \text{PotentialObject}(po) \rightarrow \\
\quad (\text{Equiv}(o, po) \leftrightarrow (\exists bc \exists m \exists pbc \exists m' \exists behaviorConstituent(bc, bct, m) \land \\
\quad \text{PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc) \land \\
\quad \text{SEObj}(o, bc, or) \land \text{SEObj}'(po, pbc, or))) \]
\[(E_3) \ \text{RelationTuple}(rt) \land \neg \text{KnownRelationTuple}(rt) \land \text{PotentialRelationTuple}(prt) \rightarrow \\
(Equiv(rt, prt) \iff \exists bc \exists m \exists pbc \exists m' \exists bct \exists rel \exists or_1, ..., \exists or_n \\
\text{BehaviorConstituent}(bc, bct, m) \land \\
\text{PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc) \land \\
\text{SERelTuple}_{n}(rt, bc, rel, or_1, ..., or_n))
\]

The definition of the helper predicates SCObj, SCObj', SEObj, SEObj, SERelTuple_{n}, and SERelTuple_{n} should be obvious - they identify the (potential) structural element that is the first parameter as bound to a specified role within the (potential) behavior constituent instance being the second parameter.

**Property (SE_2):** First, we show that the construction of Equiv is compatible with the structural requirements stated for behavior constituents:

\[
\text{BehaviorConstituent}(bc, bct, m) \land \text{PotentialBehaviorConstituent}(pbc, bct', m') \land \\
\text{Equiv}(bc, pbc) \Rightarrow (E_1) \\
\forall or \text{SCObj}(o, bc, or) \land \text{SCObj}'(po, bc, or) \rightarrow \text{Equiv}(o, po) \land \\
bct = bct' \Rightarrow (E_2) \\
\forall or \text{SEObj}(o, bc, or) \land \text{SEObj}'(po, pbc, or) \rightarrow \text{Equiv}(o, po) \\
q. e. d.
\]

Together, this proves that for all object roles of equivalent behavior constituents, both from the structural conditions and from the structural effects, the respective bound objects (actual or potential) are equivalent. The other direction is even simpler to show:

\[
\text{BehaviorConstituent}(bc, bct, m) \land \text{PotentialBehaviorConstituent}(pbc, bct', m') \land \\
\forall or \text{Maps}(m, or, o) \land \text{Maps}'(m', or, po) \rightarrow \text{Equiv}(o, po) \Rightarrow (by \ restricting \ to \ SC \ obj \ roles) \\
\forall or \text{SCObj}(o, bc, or) \land \text{SCObj}'(po, pbc, or) \rightarrow \text{Equiv}(o, po) \Rightarrow (E_1) \\
\text{Equiv}(bc, pbc) \quad q. \ e. \ d.
\]

Hence, (SE_2) is valid for all behavior constituent instances.

The remainder of the proof, which will show Equiv to be a well-defined injective complete mapping (properties M, I and C) obeying the remaining requirements of structural equivalence (SE_1), is constructed as an induction on the order of elements (objects, relations, and behavior constituents), as defined by

\[
\text{order}: \{e : \text{Element}(e)\} \cup \{bc : \exists bct \exists m \text{BehaviorConstituent}(bc, bct, m)) \rightarrow \kappa_0 \\
(O_1) \quad \text{order}(e) = 0 \quad \text{iff KnownElement}(e) \\
\text{order}(e) = 1 + \min(\{\text{order}(bc) : \text{IsSCElem}(e, bc)) \quad \text{otherwise} \\
(O_2) \quad \text{order}(bc) = \max(\{\text{order}(e) : \text{IsSCElem}(e, bc))
\]

Again, the definition relies on simple helper predicates, IsSCElem and IsSCElem. If there is no backward completion, the order of any element or behavior constituent is well-defined. Recall the fundamental law from section 6.2:

\[
\text{Element}(e) \rightarrow \text{KnownElement}(e) \lor \text{EffectElement}(e) \lor \text{IntroducedElement}(e)
\]

The latter alternative is excluded (\forall e \neg \text{IntroducedElement}(e)), which corresponds to the global closed-world assumption, see section 7.2.2). Hence, each structural element, e, is either from the
situation description (KnownElement(e)) or the effect of a behavior constituent instance (EffectElement(e)). The initial situation description is assigned order 0 and all parts of the forward completion are created by a finite set of behavior constituents. The order of a structural element represents the minimal number of behavior constituents necessary - so potential loops in the creation structure do not affect the order.

**Lemma 1:** Before beginning the induction, we start with a simple lemma:

\[(\text{Lemma 1}) \quad \text{RelationTuple}_n(rt, rel, o_1, ..., o_n) \land \text{order}(rt) = n+1 \rightarrow \text{order}(o_1) \leq n+1 \land ... \land \text{order}(o_n) \leq n+1\]

**Proof:**

\[ \text{RelationTuple}_n(rt, rel, o_1, ..., o_n) \land \text{order}(rt) = n+1 \Rightarrow (O_1) \]
\[ \exists bc \exists bct \exists m \quad \text{BehaviorConstituent}(bc, bct, m) \land \text{IsSEElem}(rt, bc) \land \text{order}(bc) = n \Rightarrow (\text{definition of structural effects}) \]
\[ \exists o_1 ... \exists o_n (\text{SCObj}(o_1, bc, or_1) \lor \text{SEObj}(o_1, bc, or_1)) \land ... \land (\text{SCObj}(o_n, bc, or_n) \lor \text{SEObj}(o_n, bc, or_n)) \Rightarrow (O_j) \]
\[ \text{order}(o_1) \leq n+1 \land ... \land \text{order}(o_n) \leq n+1 \]

The base case, \(\text{order}(x) = 0\), turns out to be quite simple. In fact, for structural elements, it is trivial, since the properties (C), (M), (I), and (SE1) can all be seen from the definition of the starting point and (E0). For behavior constituents of order 0, we have to show that equivalent potential ones exist (property (C)) as follows:

\[\forall e \quad \text{SCElem}(e, bc) \rightarrow \text{order}(e) = 0 \Rightarrow (\text{base case for property C})\]
\[\forall e \quad \text{SCElem}(e, bc) \rightarrow \exists pe \text{ PotentialElement}(pe) \land \text{Equiv}(e, pe) \Rightarrow (\text{by appropriate construction})\]
\[\exists m' \text{ PotentialSC}(bct, m') \Rightarrow (IR_1 \text{ (first instantiation rule)})\]
\[\exists pbc \exists m'' \text{ Extends}(m'', m', bct) \land \text{PotentialBehaviorConstituent}(pbc, bct, m'') \Rightarrow (E_1: \text{ SC elements equivalent})\]
\[\text{Equiv}(bc, pbc) \]

Hereby, the base case is complete.

For the general case \(\text{order}(x) > 0\), slightly more effort is required. We will start with properties (M) and (I), followed by (SE1) and, finally, the central claim (C) will be shown.

**Property (M):** The proof that \text{Equiv} is a well-defined mapping relies on the unification rules (UR1) and (UR2):

\[\text{Object}(o) \land \text{Equiv}(o, po) \land \text{Equiv}(o, po') \land \text{order}(o) = n+1 \Rightarrow (O_j)\]
\[\exists bc \exists bct \exists m \exists or \quad \text{BehaviorConstituent}(bc, bct, m) \land \text{SEObj}(o, bc, or) \land \text{order}(bc) = n \Rightarrow (\text{induction assumption (C)})\]
\[\exists pbc \exists m' \text{ PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc) \land \exists po'' \text{ SEObj}'(po'', pbc, or) \Rightarrow (UR_2)\]
\[po = po'' \land po' = po'' \]

q. e. d.
The proof of property (M) for relation tuples is mainly analogous:

\[
\text{RelationTuple}_n(r,t, \text{rel}, o_1, ..., o_n) \land \text{PotentialRelationTuple}_n(p,t, \text{rel}, p_{o_1}, ..., p_{o_n}) \land \\
\text{Equiv}(r,t) \land \text{Equiv}(p,t') \land \text{order}(r) = n+1 \\
\Rightarrow \quad (O_I)
\]

\[
\exists bc \exists bct \exists m \exists r1 ... \exists on, \text{BehaviorConstituent}(bc, bct, m) \land \\
\text{SERelTuple}_n(r,t, bc, r, \text{rel}, o_1, ..., o_n) \land \\
\text{SERelTuple}_n(p,t, bc, r, \text{rel}, o_1, ..., o_n) \land \\
\Rightarrow \quad \text{(induction assumption (C))}
\]

\[
\exists pc \exists m' \exists bct \exists m'' \exists r1 ... \exists on', \text{BehaviorConstituent}(bc, bct, m) \land \\
\exists pc' \exists m'' \exists bct' \exists m''' \exists r1 ... \exists on', \text{BehaviorConstituent}(bc', bct', m') \land \\
\Rightarrow \quad \text{(definition of structural effects)}
\]

\[
\text{BehaviorConstituent}(bc, bct, m) \land \text{Equiv}(bc, pbc) \land \\
\Rightarrow \quad \text{(UR2)}
\]

For behavior constituents this reads

\[
\text{BehaviorConstituent}(bc, bct, m) \land \\
\Rightarrow \quad (O_2)
\]

\[
\forall e \text{IsSCElem}(e, bc) \rightarrow \text{order}(e) \leq n \Rightarrow \quad (E_I \text{ applied to equivalences})
\]

\[
\forall o \text{SCObj}(o, bc, or) \land \text{SCObj}'(po, pbc, or) \land \text{SCObj}'(po', pbc', or) \rightarrow \\
\Rightarrow \quad \text{(induction assumption (M))}
\]

\[
\forall o \text{SCObj}'(po, pbc, or) \land \text{SCObj}'(po', pbc', or) \rightarrow \text{po} = \text{po}' \Rightarrow \quad \text{(UR1)}
\]

\[
pbc = pbc' \quad \text{q. e. d.}
\]

Property (I): That \text{Equiv} is an injection can be seen as follows

\[
\text{Object}(o) \land \text{Equiv}(o, po) \land \text{Equiv}(o', po) \land \text{order}(o) = n+1 \Rightarrow \quad (E_2)
\]

\[
\exists bc \exists bct \exists m \exists po \exists r1 ... \exists on, \text{BehaviorConstituent}(bc, bct, m) \land \\
\Rightarrow \quad \text{(UR2)}
\]

\[
\text{BehaviorConstituent}(bc, bct, m) \land \text{Equiv}(bc, pbc) \land \text{Equiv}(bc', pbc) \land \\
\Rightarrow \quad \text{(property I for objects)}
\]

\[
\text{BehaviorConstituent}(bc, bct, m) \land \\
\Rightarrow \quad \text{(RelationTuple)}
\]

For behavior constituents we get

\[
\text{BehaviorConstituent}(bc, bct, m) \land \text{BehaviorConstituent}(bc', bct, m') \land \\
\Rightarrow \quad (O_2)
\]

\[
\forall e \text{IsSCElem}(e, bc) \lor \text{IsSCElem}(e, bc') \rightarrow \text{order}(e) \leq n \Rightarrow \quad (E_I \text{ applied to equivalences})
\]
∀\text{or SCObj}(o, bc, or) \land \text{SCObj}(o', bc', or) \land \text{SCObj}'(po, pbc, or) \rightarrow 
\text{Equiv}(o, po) \land \text{Equiv}(o', po) \Rightarrow (\text{induction assumption (I)})

∀\text{or SCObj}(o, bc, or) \land \text{SCObj}(o', bc', or) \rightarrow o = o' \Rightarrow (UR_1)

bc = bc'
q. e. d.

Property (SE_1): The structural equivalence requirement w. t. relation tuples follows from

\text{RelationTuplen}(rt, rel, o_1, ..., o_n) \land \text{PotentialRelationTuplen}(prt, rel', o_1, ..., o_m) \land 
\text{Equiv}(rt, prt) \land \text{order}(rt) = n+1 \Rightarrow (O_1)

∃bc ∃bct ∃m ∃or_1 ... ∃or_n \text{BehaviorConstituent}(bc, bct, m) \land 
\text{SERelTuplen}(rt, bc, rel, or_1, ..., or_n) \land \text{order}(bc) = n \Rightarrow (\text{induction assumption (C)})

∃pbc ∃m' \text{PotentialBehaviorConstituent}(pbc, bct, m') \land 
\text{Equiv}(bc, pbc) \land 
∃po_1' ... ∃po_n' \text{Maps}(m, or_1, o_1) \land ... \land \text{Maps}(m, or_n, o_n) \land 
\text{Maps}(m', or_1, po_1') \land ... \land \text{Maps}(m', or_n, po_n') \Rightarrow (\text{definition of structural effects})

∃prt' \text{PotentialRelationTuplen}(prt', rel, po_1', ..., po_n') \land \text{Equiv}(rt, prt') \Rightarrow (\text{property M})

prt = prt' \Rightarrow (\text{PotentialRelationTuplen})

po_1 = po_1' \land ... \land po_n = po_n' \Rightarrow (SE_2)

Equiv(o_1, po_1) \land ... \land \text{Equiv}(o_n, po_n) \Rightarrow (\text{RelationTuplen})
q. e. d.

And for the other implication direction:

\text{RelationTuplen}(rt, rel, o_1, ..., o_n) \land \text{PotentialRelationTuplen}(prt, rel, po_1, ..., po_n) \land 
\text{Equiv}(o_1, po_1) \land ... \land \text{Equiv}(o_n, po_n) \land \text{order}(rt) = n+1 \Rightarrow (O_1)

∃bc ∃bct ∃m ∃or_1 ... ∃or_n \text{BehaviorConstituent}(bc, bct, m) \land 
\text{SERelTuplen}(rt, bc, rel, or_1, ..., or_n) \land \text{order}(bc) = n \Rightarrow (\text{induction assumption (C)})

∃pbc ∃m' \text{PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc) \land 
∃po_1' ... ∃po_n' \text{Maps}(po_1', pbc, or_1) \land ... \land \text{Maps}(po_n', pbc, or_n) \Rightarrow (SE_2)

Equiv(rt, prt') \land \text{Equiv}(o_1, po_1') \land ... \land \text{Equiv}(o_n, po_n') \Rightarrow (M)

po_1 = po_1' \land ... \land po_n = po_n' \Rightarrow (\text{RelationTuplen})
prt = prt' \Rightarrow (\text{RelationTuplen})
q. e. d.

Property (C): The central claim of theorem 1, expressed as the completeness of the \text{Equiv} mapping, is shown for objects first - since this will be used for behavior constitutents of the same order. For each object, there is an equivalent potential object:

\text{Object}(o) \land \text{order}(o) = n+1 \Rightarrow (O_1)

∃bc ∃bct ∃m ∃or \text{BehaviorConstituent}(bc, bct, m) \land \text{SEObj}(o, bc, or) \land 
\text{order}(bc) = n \Rightarrow (\text{induction assumption (C)})

∃pbc ∃m' \text{PotentialBehaviorConstituent}(pbc, bct, m') \land \text{Equiv}(bc, pbc) \land 
∃po \text{SEObj}'(po, pbc, or) \Rightarrow (E_2)

\text{Equiv}(o, po) \Rightarrow (E_2)
q. e. d.

For relation tuples:

\text{RelationTuple}(rt) \land \text{order}(rt) = n+1 \Rightarrow (O_1)

∃bc ∃bct ∃rel ∃or_1 ... ∃or_n \text{BehaviorConstituent}(bc, bct, m) \land \text{SERelTuple}(rt, bc, rel, or_1, ..., or_n) \land 
\text{order}(bc) = n \Rightarrow (\text{induction assumption (C)})
∃pbc ∃m' PotentialBehaviorConstituent(pbc, bct, m') ∧ Equiv(bc, pbc) ∧
∃prt SERelTuple_n(prt, pbc, rel, or1, ..., orn) ⇒ (E2)
Equiv(rt, prt) q. e. d.

Finally, we can proceed to prove that also each behavior constituent instance possesses an equivalent potential one:

BehaviorConstituent(bc, bct, m) ∧ order(bc) = n ⇒ (O2)
∀e IsSCElement(e, bc) ⇒ order(e) ≤ n ⇒ (induction assumption (C))
∀e IsSCElement(e, bc) ∃pe PotentialElement(e) ∧ Equiv(e, pe) ⇒ (by appropriate construction)
∃m' PotentialElement(e) ∧ Equiv(e, pe) ⇒ (IR1 (first instantiation rule))
∃pbc ∃m" Extends(m", m', bct) ∧
PotentialBehaviorConstituent(pbc, bct, m") ⇒ (E1: SC elements equivalent)
Equiv(bc, pbc) q. e. d.

Together, this constitutes a proof of theorem 1.

Theorem 2 states that this structurally equivalent subset identified in theorem 1 will identified as "active" and "existing" by the activity rules (AR1) to (AR4) together with the starting point of labeling all potential elements that are equivalent to the initial situation description "existing”:

KnownElement(e) ∧ Equiv(e, pe) ⇒ exists(pe)

Again, we will repeat the theorem before proceeding with the proof.

**Theorem 2:**

If a potential behavior constituent or a potential structural element is part of the structurally equivalent subset defined in theorem 1 (i.e. it has an actual counterpart), it is active or "existing", respectively. Formally:

PotentialBehaviorConstituent(pbc, bct, m) →
(∃bc ∃m' BehaviorConstituent(bc, bct, m') ∧ Equiv(bc, pbc) → active(pbc))

PotentialElement(pe) →
(∃e Element(e) ∧ Equiv(e, pe) → exists(pe))

**Proof:**

The proof relies on the order of (actual) elements and behavior constituents, as defined above ((O1) and (O2)). Again, induction on this order will be used.

The base case is trivial and can be seen from the starting point (see above) and (O1).

For the general case, we start with the potential behavior constituents:

PotentialBehaviorConstituent(pbc, bct, m) ∧ BehaviorConstituent(bc, bct, m') ∧
Equiv(bc, pbc) ∧ order'(bc) = n ⇒ (SE2)
∀pe IsSCElement(pe, pbc) ⇒ ∃e IsSCElement(e, bc) ∧ Equiv(e, pe) ⇒ (O2)
∀pe IsSCElement(pe, pbc) ⇒ ∃e Equiv(e, pe) ∧ order'(e) ≤ n ⇒ (by induction assumption)
∀pe IsSCElement(pe, pbc) ⇒ exists(pe) ⇒ (AR1)
exists(pbc) ⇒ (OR1 (for bc))
QC(bct, m') ⇒ (by definition of QC')
QC'(bct, m) ⇒ (AR2)
active(pbc) q. e. d.
For structural elements, we have
\[
\text{PotentialElement}(pe) \land \exists e \text{ Element}(e) \land \text{Equiv}(e, \text{pe}) \land \
\text{order}(e) = n+1 \Rightarrow (O_1)
\]
\[
\exists bc \exists bct \exists m \text{ BehaviorConstituent}(bc, bct, m) \land \text{IsSEElem}(e, bc) \land \
\text{order}(bc) = n \Rightarrow (C, \text{ i. e. theorem 1})
\]
\[
\exists pbc \exists m' \text{ PotentialBehaviorConstituent}(pbc, bct, m) \land \
\text{Equiv}(bc, pbc) \Rightarrow (SE_2)
\]
\[
\exists pe' \text{ IsSEElem}'(pe', pbc) \land \text{Equiv}(e, pe') \Rightarrow (M)
\]
\[
\text{pe} = \text{pe}' \Rightarrow (\text{summarizing})
\]
\[
\exists bc \exists pbc \text{ Equiv}(bc, pbc) \land \text{IsSEElem}'(pe, pbc) \land \text{order}(bc) = n \Rightarrow (\text{induction assumption})
\]
\[
\text{active}(pbc) \Rightarrow (AR_3)
\]
\[
\exists \text{pe} = \text{pe}' \Rightarrow (\text{summarizing})
\]
\[
\exists bc \exists pbc \text{ Equiv}(bc, pbc) \land \text{IsSEElem}'(pe, pbc) \land \text{order}(bc) = n \Rightarrow (\text{induction assumption})
\]

This constitutes a proof of theorem 2.

Finally, theorem 2b, which yields stronger results for loop-free creation structures, will be proven:

**Theorem 2b:**

If there are no loops in the creation structure, then potential behavior constituents are active and potential structural elements exist, if and only if they are part of the structurally equivalent subset defined by theorem 1 (i.e., they have actual counterparts). Formally:

\[
\text{PotentialBehaviorConstituent}(pbc, bct, m) \rightarrow \exists bc \exists m' \text{ BehaviorConstituent}(bc, bct, m') \land \text{Equiv}(bc, pbc) \leftrightarrow \text{active}(pbc)
\]

\[
\text{PotentialElement}(pe) \rightarrow \exists e \text{ Element}(e) \land \text{Equiv}(e, pe) \leftrightarrow \text{exists}(pe)
\]

**Proof:**

Once more, the proof is organized as an induction on the creation order, but this time defined for potential elements and in a different way. Since we rely on the absence of loops in the creation structure, we can define the order of potential elements and behavior constituents as the maximum length of a creation path. order' is defined as the minimal mapping satisfying the following laws:

\[
\text{order'}: \{\text{pe}: \text{PotentialElement}(pe)\} \cup \{\text{pbc}: \exists bc \exists m \text{ PotentialBehaviorConstituent}(bc, bct, m)\} \rightarrow \mathbb{N}
\]

\[
(O_1') \quad \text{order'}(pe) = 0 \quad \text{iff } \exists e \text{ KnownElement}(e) \land \text{Equiv}(e, pe)
\]
\[
\text{order'}(pe) = 1 + \max(\{\text{order'}(pbc) : \text{IsSEElem}'(pe, pbc)\}) \quad \text{otherwise}
\]

\[
(O_2') \quad \text{order'}(pbc) = \max(\{\text{order'}(pe) : \text{IsSCElem}'(pe, pbc)\})
\]

As a remark, we can rely on all "creation paths" to start in the initial situation description, so there is a single consistent definition of order'.

The base case will be skipped again (see the starting point and (O_1')). The only implications left to prove are the "backward" directions of theorem 2b, since the "forward" direction can be seen from the proof of theorem 2 already.
We start with the general case for potential behavior constituents:

\[ \text{PotentialBehaviorConstituent}(pbc, bct, m) \land \text{active}(pbc) \implies (AR_2) \]
\[ \exists pbc \land QC'(bct, m) \implies (AR_1) \]
\[ \forall p \in \text{SCElem}(p, pbc) \rightarrow \exists p \implies (O_1') \]
\[ \forall p \in \text{SCElem}(p, pbc) \rightarrow \exists p \land \text{order}'(p) \leq n \implies (\text{induction assumption}) \]
\[ \exists m' \in \text{SC}(bct, m') \land QC(bct, m') \implies (OR_1) \]
\[ \exists bc \exists m'' \in \text{Extends}(m'', m', bct) \land \text{BehaviorConstituent}(bc, bct, m'') \land \]
\[ (\forall o \in \text{SCO}(o, bc, or) \land \text{SCO}'(po, pbc, or) \rightarrow \]
\[ \text{Equiv}(o, po)) \implies (E_1) \]
\[ \exists bc \land \text{Equiv}(bc, pbc) \]

For structural elements the proof relies on the additional rule (AR_{3b}):

\[ \text{PotentialElement}(p) \land \exists p \land \text{order}'(p) = n+1 \implies (AR_{3b}) \]
\[ \exists pbc \exists bct \exists m \in \text{PotentialBehaviorConstituent}(pbc, bct, m) \land \text{active}(pbc) \land \]
\[ \text{IsSEElem}'(p, pbc) \implies (O_2' \text{ (order' as maximum!}) \]
\[ \text{order}'(pbc) \leq n \implies (\text{induction assumption bc}) \]
\[ \exists bc \exists m' \in \text{BehaviorConstituent}(bc, bct, m') \land \text{Equiv}(bc, pbc) \implies (SE_2) \]
\[ \exists e \in \text{Element}(e) \land \text{IsSEElem}(e, bc) \land \text{Equiv}(e, p) \quad q. e. d. \]

Together, this constitutes a proof of theorem 2.

Thus, the correct set of behavior constituents is activated in the extended forward space.
References


