ELSEVIER

Artificial Intelligence 117 (2000) 173–229

Artificial Intelligence

www.elsevier.com/locate/artint

Model-based reasoning about learner behaviour

Kees de Koning, Bert Bredeweg *, Joost Breuker, Bob Wielinga

Department of Social Science Informatics (SWI), University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, Netherlands

Received 23 August 1999

Abstract

Automated handling of tutoring and training functions in educational systems requires the availability of articulate domain models. In this article we further develop the application of qualitative models for this purpose. A framework is presented that defines a key role for qualitative models as interactive simulations of the subject matter. Within this framework our research focuses on automating the diagnosis of learner behaviour. We show how a qualitative simulation model of the subject matter can be reformulated to fit the requirements of general diagnostic engines such as GDE. It turns out that, due to the specific characteristics of such models, additional structuring is required to produce useful diagnostic results. A set of procedures is presented that automatically maps detailed simulation models into a hierarchy of aggregated models by hiding non-essential details and chunking chains of causal dependencies. The result is a highly structured subject matter model that enables the diagnosis of learner behaviour by means of an adapted version of the GDE algorithm. An experiment has been conducted that shows the viability of the approach taken, i.e., given the output of a qualitative simulator the procedures we have developed automatically generate a structured subject matter model and subsequently use this model to successfully diagnoses learner behaviour.

Keywords: Qualitative reasoning; Model aggregation; Model-based diagnosis; Student modelling; Interactive learning environments

1. Introduction

Both in professional and every-day life people have to interact with and reason about a large number of systems. Computer simulations can be used to construct interactive environments by means of which people can develop knowledge about the behaviour of these systems. The steady increase in computing power has in fact given simulation a

* Corresponding author. Email: bert@swi.psy.uva.nl.

0004-3702/00/$ – see front matter
PII: S0004-3702(99)00106-X

solid position within the area of educational systems [26]. However, several studies have shown that simulations are only effective when proper guidance is provided [36,51,67, 69]. Automating certain tutoring and training functions in order to provide such guidance requires the simulation model to be *articulate* [12,39,41]. Two further requirements follow from this. Firstly, a specific simulation model should manifest all the behavioural features of the 'real' system as far as those are relevant to the educational goals. Secondly, a simulation model should contain the appropriate *handles*, by means of which these features are indexed, to enable a knowledgeable communication with the learner about the model contents. Qualitative simulators, such as QPE [39] and GARP [8], provide a basis for generating articulate simulation models. Given such models, a remaining challenge concerns the automated handling of guidance by the educational system.

Providing guidance means that the learning environment should be able to adapt the interaction to the situation at hand, both with respect to the specific subject matter that is considered and to the individual learner it is interacting with. Guidance may take on many forms, such as providing explanations, presenting counter examples, suggesting assignments, and the like. Whatever the specific form, individual guidance requires knowledge about the learner and hence involves assessment of his or her knowledge. More specifically, the educational system has to diagnose the learner's *problem solving behaviour*.[1] The subject matter we are dealing with in this article concerns 'behaviour analysis', i.e., learners should acquire problem solving skills such as predicting or 'postdicting' (explaining) the behaviour of systems using qualitative terms. Hence, the learner's problem solving behaviour consists of a set of inferences about the behaviour of these systems. The way the student interacts with the learning environment reflects this problem solving behaviour. The educational system therefore has to monitor this interaction (performance assessment) and diagnose deviations, with respect to some norm, in terms of problem solving errors made by the learner (behaviour diagnosis).

In research on artificial intelligence in education, diagnosis of problem solving behaviour is generally considered extremely difficult, or even infeasible [75]. One important reason is that behaviour diagnosis is usually based on hand-crafted catalogues of misconceptions, or bugs, which makes diagnostic systems very expensive to develop. Moreover, a catalogue of bugs is only applicable to one specific domain. When the subject matter changes, a new catalogue has to be developed and implemented. Alternative approaches that try to overcome these difficulties, e.g., by generating bugs dynamically [18], have also proven to be difficult. Although no hand-crafted catalogues are needed, domain specific filters have to be used to avoid generation of implausible bugs. In addition, the construction of generative theories has turned out to be problematic, mainly as a result of lacking operational theories on how people acquire knowledge.

In this article we propose a different, *model-based*, approach [47] to diagnose learner behaviour. Following [62], this choice is based on the fact that model-based diagnosis is an extensively studied and well-understood field of research, and hence we can reuse existing techniques and representations. Model-based diagnosis claims to be generic and domain

[1] Throughout this article, the term 'learner behaviour' refers to the 'problem solving behaviour of a learner'. 'Learner behaviour' should not be confused with 'system behaviour' which refers to the subject matter that the student has to master.

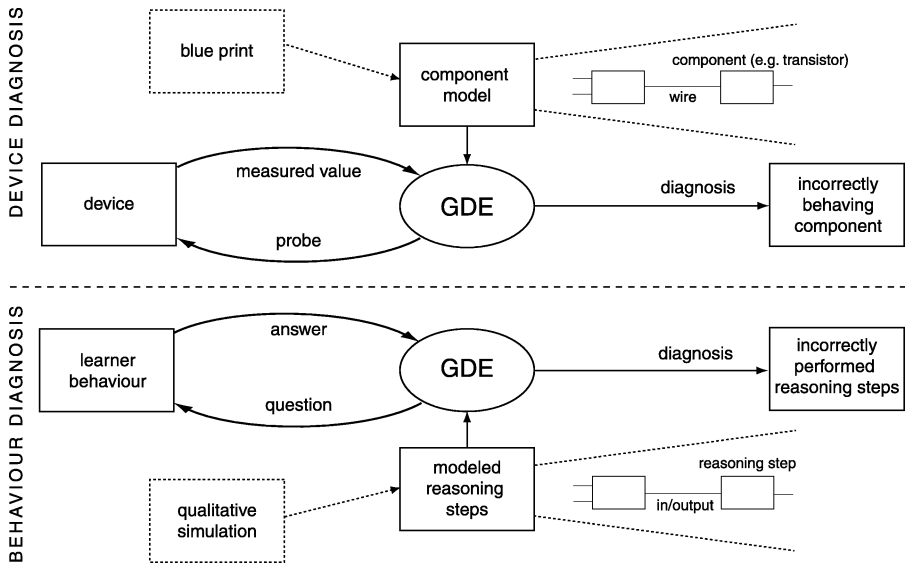

Fig. 1. Model-based diagnosis of device versus learner behaviour.

independent. It does not require explicit fault models or bug catalogues for its operation, but instead reasons from a representation of the correct behaviour of the system to be diagnosed. More specific, in model-based diagnosis the behaviour of a device is compared to the behaviour predicted by a model of that device. In the case of diagnosing learner behaviour we propose to automatically generate this model using a qualitative simulator. On the basis of the output of the simulator a subject matter model can be constructed that acts as a normative model of the learner behaviour and as such can be used for model based diagnosis. Deviations in the problem solving behaviour of the learner, i.e., incorrect predictions or postdictions concerning the behaviour of the system that is the subject of the teaching activity, are then diagnosed as inferences made by the qualitative simulator that the learner cannot have applied correctly given the observations. Fig. 1 illustrates the high-level mapping between model-based diagnosis of device behaviour and learner behaviour.

The same diagnostic component (e.g., GDE [30]) can be applied to different (subject matter) models, as long as they comply with the modelling principles required by the diagnostic algorithms. One of the main requirements is that the system under diagnosis can be modelled as a set of *connected components*, for which the behaviour can be defined individually. These components should model the smallest entities that can be individually repaired; diagnosis is only useful down to the level of possible repairs. In the context of learning, this means that these components should represent the smallest units of problem solving knowledge that are still relevant in an educational setting (e.g., that can be individually explained). The constraints that the component-connection paradigm puts on the subject matter models are not only a requirement for diagnosis, but are also useful for other educational functions: a clear separation of the knowledge in terms of indexed elements that make sense in education is a prerequisite for automated structured

explanation, generation of questions, or subject matter sequencing. Only with such a modular structure, different components can be freely combined, selected, and sequenced.

Following the above argumentation, we propose a framework for the construction of interactive learning environments based on qualitative models. These models are used as simulations of the subject matter and provide the basis for the construction of advanced educational functions to support the interaction with the learner. The procedures implementing this approach should be domain independent, i.e., they should not depend on the specific system that is being simulated by the qualitative simulator, and they should do their work fully automatic. In the research presented we focus on the function 'diagnosis of learner behaviour' because it is central to any form of individualised interaction.

The following topics must be addressed. First, it should be shown that the qualitative simulator represents knowledge that fits the way teachers and learners communicate about system behaviour. The qualitative simulator should produce a description of the system behaviour that is useful in an educational setting, i.e., it should be *didactically plausible*. It should produce a description that learners can comprehend and in principle learn. Although important, the construction of such articulate models is not the main focus of the research presented here. Section 2.3 reports on previously published research [33] that supports the hypothesis that the simulator we use produces such models.

The second topic concerns the mapping of the simulation model onto the component-connection paradigm (Section 3). Recall that the simulation model reflects all the necessary inference steps that must be made in order to produce a correct behaviour analysis. Thus, the simulation model provides the ingredients for the norm model of the model-based diagnostic engine and hence all the inferences must be represented as 'inference components' in this norm model. Typically, each inference should be represented as an individual, context independent inference component that derives output from input using a behaviour rule. In addition, behaviour rules should be defined for deriving input from output. Being able to construct a component-connection model out of the simulation model means that we can represent the skill 'qualitative behaviour analysis' as the norm model to be used by the diagnostic engine. Hence, we are able to diagnose learner behaviour in terms of how it deviates from this model.

The third topic concerns the tractability of the general diagnostic engine (Section 4). It turns out that due to the specific characteristics of qualitative simulation models, additional structuring is required in order to deliver useful diagnostic results. One reason is that the component's behaviour rules make use of qualitative calculi, which are relatively weak compared to the behaviour rules used in typical applications of model-based diagnosis such as electronic circuits. In artifacts, solving the tractability problem can be done by using the hierarchical structure that is often evident from the physical or functional structure of the device. Such a natural decomposition is not available for the kind of norm models that we use. The inference components that are part of this model do not necessarily map onto components in the 'real' system. Alternative structuring principles have to be developed for these models. As mentioned before, an additional requirement is that the structuring must be performed automatically on the basis of the component-connection model that has been generated using the output of the simulator.

A fourth and related topic concerns the fact that the diagnostic engine has to operate in the context of an educational system. Measurement selection, and candidate discrimination

in general, are bound to the ongoing discourse between the education system and the learner. Putting the diagnostic engine to work in the context of an educational system is subject of Section 5. In the next section (Section 2) we will first discuss the theoretical background of our approach.

2. Theoretical background

Important to the research presented in this article is the idea of using a qualitative simulation model as the kernel of an educational system. Section 2.1 describes the main characteristics of the simulator that we use for this purpose. How the qualitative simulator can be embedded in an interactive learning environment is subject of Section 2.2, which presents an architecture for simulation-based educational systems. Putting the qualitative simulation in the heart of the educational system puts specific requirements on the simulator. Section 2.3 discusses research that supports the hypothesis that the simulator we use produces didactically plausible models. Finally, Section 2.4 discusses the role of diagnosis in educational systems. We emphasise the importance of 'behaviour diagnosis', in contrast to 'learner modelling', and the usefulness of model-based techniques for performing the former diagnostic task.

2.1. Generating articulate simulation models

Qualitative simulation models typically represent knowledge about the structure and the qualitative distinct behavioural features of a system. To produce such models we use GARP [8], an interactive simulator that predicts behaviour starting from a user selected scenario. The scenario captures the initial structural description of a system, possibly augmented with some initial behavioural facts. GARP can be controlled to produce a full simulation at once, or to work on specific states of behaviour in interaction with the user. Model fragment libraries [37] are used to feed the simulator with domain specific knowledge. The simulator itself implements a domain independent qualitative reasoning shell. [2]

The primitives that can be used within the model fragments for representing the behaviour of a system include: quantities, values, behavioural constraints, and causal dependencies. Quantities represent the changing properties of the system. Values are represented in quantity spaces (Qsp), i.e., ordered sets of alternating points and intervals. Values are assigned to quantities using a dedicated predicate. Values from different quantity spaces are considered unrelated except when these quantity spaces include the value 'zero', which is universal to all quantity spaces. In such cases, simple transitive inequality reasoning is in principle possible, e.g., $\mathsf{value}(Q_1, V_1)$, $\mathsf{value}(Q_2, V_2)$, $\mathsf{Qsp1}: V_1 < 0$, $\mathsf{Qsp2}: V_2 > 0 \rightarrow Q_1 < Q_2$. Behavioural constraints are implemented by inequality statements and serve a number of purposes. They can be used to specify a range of values that a quantity may have (e.g., $\mathsf{temperature1} > \mathsf{freezingpoint1}$), they can be used for specifying dependencies between quantities (e.g., $\mathsf{temperature1} > \mathsf{temperature2}$), and

[2] GARP is freeware and implemented in SWI-Prolog [77].

they can be use to relate the values of different quantity spaces (e.g., freezingpoint1 > freezingpoint2).

In a specific state a quantity has a value, or its value is unknown. In addition, behavioural constraints may hold. If a quantity does not have a value and is not involved in any constraints it obviously has no effect on the simulation and can be considered superfluous in that state of behaviour. When the value refers to an interval, notice the subtle difference between 'having the same value' and 'being equal'. Quantities may have the same qualitative interval value and at the same time be unequal. For example, two similar containers filled with different amounts of liquid, may be represented as value(Level1, plus) and value(Level2, plus) and Level1 > Level2.

Knowledge about causality is represented using an adapted version of QPT [38]. Changes are 'caused by' influences as defined by *processes* and *agents*. The former represent flows that occur because entities differ on some quantity, for example, a flow of energy between two objects because they differ in temperature. Agents represent external factors, e.g., someone controlling a device. Changes are propagated to other quantities by proportionalities. Correspondences can be used to relate certain quantity values. However, the notion of correspondence in GARP is different from the one used in QPT. First, correspondences are defined to be directed or undirected, referring to a causal dependency or to a non-causal one. Second, correspondences are defined for either specific values of two quantities (*value correspondence*), or for all the values that two quantities can have (*quantity space correspondence*). In the case of the latter, such quantities have the same type of quantity space.

Qualitatively distinct states of behaviour are characterised by differences in the set of quantity values and/or inequality statements that hold in each state. *Termination rules* implement knowledge about state changes. Two sets of termination rules exist. One set deals with individual quantities. These rules specify the value that a quantity will have in the next state, given the value it has in the current state and the sign of its derivative (i.e., its direction of change). The limit rule [28] is an example of such a rule. The second set of rules deals with inequalities between quantities. They specify how inequality statements in the current state will change, given the signs of the derivatives of the involved quantities. For example, two quantities may become unequal because one of them increases whereas the other remains constant. Not all potential changes will lead to actual state changes. Some will have to be ignored because other changes precede them. In addition, changes may be merged because they are related. Knowledge on these issues is implemented by *precedence rules*. The simulator uses these rules to merge related changes first. It then removes those sets of changes that are preceded by other sets. The remaining sets of changes are (by definition) fully independent. State changes are therefore generated for each individual set and for all possible combinations of these sets. Some quantities have derivatives with sign 'zero' and may therefore not be involved in any change. *Continuity rules* are used to explicitly specify non-changing quantities between states.

2.2. Architecture for learning environments

Fig. 2 illustrates how a qualitative simulator can be embedded in an interactive learning environment. The figure presents an abstract view on the interaction between the learner

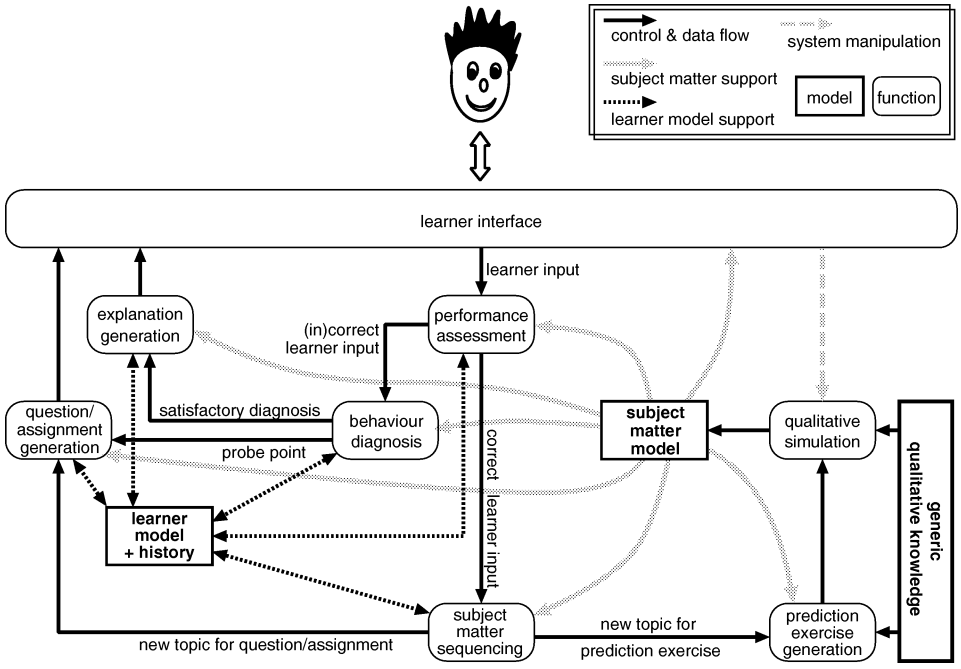

Fig. 2. Conceptual architecture for an educational system.

and the simulation, showing the most important tutoring and training functions (functional units). Exactly how a learning environment presents itself to a learner depends on the realisation of these functional units (see [13] and [80] for general discussions on architectures for educational systems). Below we discuss the specific implementation of the architecture as used for the construction of STAR^{light}; the prototype educational system that we have developed for evaluation purposes (Section 6).

In Fig. 2 the 'generic qualitative knowledge' refers to the library of model fragments, the different sets of rules for determining state changes, and the scenario's that can be used to generate new prediction exercises. The output of the qualitative simulator forms the basis for the subject matter model. By manipulating the interface objects the learner can control the simulator, give answers to questions or ask for help. Two main control flow loops exist: as long as the learner's behaviour is consistent with the subject matter model, the main loop is learner interface $\rightarrow$ performance assessment $\rightarrow$ subject matter sequencing $\rightarrow$ question/assignment generation. When all the topics related to a specific prediction exercise have been dealt with sufficiently, the subject matter sequencing can switch to a new prediction exercise (after which the previous loop continues). When a discrepancy is detected between the learner's behaviour and the subject matter model, the diagnoser is activated, and the main loop becomes learner interface $\rightarrow$ performance assessment $\rightarrow$ behaviour diagnosis $\rightarrow$ question/assignment generation. This loop continues until the diagnosis is satisfactory (e.g., a single deviating inference step is found); if this is the case, the diagnostic process ends, and an explanation is generated on the basis of the

diagnosis. The learner model may provide additional means for keeping track of learner specific information to further refine the activities performed by the functional units.

As shown in Fig. 2, a central role is assigned to the output that is generated by the simulator. This subject matter model is input to all the functional units in the educational system, as shown by the grey arrows. An important requirement for our research approach is to ensure that the different training and tutoring functions in the educational system operate domain independently; they must work independent of the specific system that is being simulated by the qualitative simulator. This means that the output of the simulator, i.e., the subject matter model, must consist of two intertwined but still well distinguishable parts. The first part consists of all the facts that together represent the behaviour of the system that is being simulated, i.e., the simulation model itself (e.g., volume $>$ 0 & increasing, etc.). The second part implements a kind of meta-level view [70] on the contents of this model. It consists of the qualitative vocabulary, as discussed in Section 2.1, by means of which the domain specific facts are indexed (e.g., 'volume' is a quantity, '$>$' is an in-equality statement, '0' is a value from Qsp1, etc., see [7,8] for details). In this way, the functional units can assess the subject matter model using domain independent terms. In fact, each functional unit should only assess the subject matter model using these domain independent terms. In addition, all the reasoning within these units should either be in terms of this vocabulary or fully independent from the simulation model all together (e.g., knowledge on discourse management [80]). Following this approach, we accomplish that the learning environment as a whole is domain independent to the extent that the simulator is domain independent. This is an important advancement.

2.3. Articulate models

When using simulation models in an educational system, these models should be *didactically plausible*. Two requirements follow from this. The model should manifest all the behavioural features of the real system as far as those are relevant to the educational goals and the model should be suitable for communication. The latter means that the model should provide *handles* (indexes) for interaction, which are needed to extract the right knowledge from the model [12,39–41]. Although qualitative reasoning has historical links to educational systems, it is not self-evident that qualitative simulation models indeed contain the right knowledge, in the right format, to support an educational interaction [10]. We therefore investigated whether the knowledge, terminology and reasoning, used in a real educational interaction could be covered by models generated by qualitative simulators, such as GARP and QPE. To this end, we conducted a *Wizard of Oz* experiment, in which a learner and a teacher communicate about a problem solving task via computer terminals [33]. Eight learners participated in the experiment, and three teachers. The learners were first year psychology students who had passed their final in physics at high school. The problem solving task was predicting the behaviour of a *balance system*, as is shown in Fig. 3. This physical system is also used as a running example throughout the article. On each side of the balance sits a container partially filled with water. The containers are equal in weight when empty, and have an equally sized outlet in the bottom. Through this outlet, the water flows out of the container, thereby decreasing the weight on that side of the balance. The flow rate of the two contained liquids can be different,

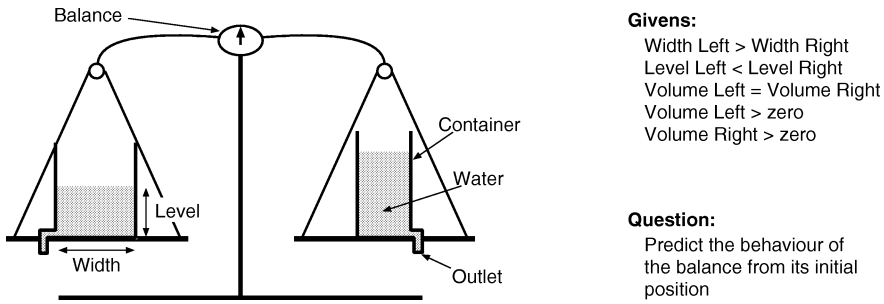

Fig. 3. The balance system.

corresponding to the pressure at the bottom. As a consequence, the balance moves to new positions, but the final state is always an equilibrium. In the experiment, a teacher and a learner discuss the behaviour of the balance system once the outlets are opened.

The analysis of the dialogue protocols showed that the terminology as it is used in the interaction is sufficiently covered by the simulation models: all concepts that appear in the protocols can be mapped to counterparts in a simulation model. This is similar to the results reported in [11]. However, the reasoning patterns employed by human reasoners differ substantially from the simulator's. One reason is that qualitative simulators usually generate all derivations in a breadth-first way; people, on the other hand, use heuristic, focused search based on "best-first" strategies [32]. A second difference is in the *grain size* of the reasoning steps taken: the inferences made by the simulators do not always map directly to the reasoning steps made by learners. For instance, a reasoning step like "the pressure is higher on the left, so the outflow is higher there as well" (*inequality proportionality*) is at a higher level of aggregation than those produced by the simulators. Nevertheless, the analysis of the dialogue protocols showed that this level of inference is considered primitive (i.e., the lowest useful level of detail) both by learners and teachers.

The analysis of the dialogue protocols has resulted in a vocabulary and a set of basic inferences (see also Table 1, Section 3) that can be used to model the way teachers and learners perform behaviour prediction. The vocabulary consists of a set of semi-formal *expressions* about entities, quantities, values, etc., that map onto the representational primitives of the qualitative simulator GARP. The basic inferences do not have direct counterparts in the simulator's output, mainly because they are at a different grain size level. The expressions and basic inferences form the basis for the definition (and the generation) of our subject matter models, which is further discussed in Section 3. For a detailed discussion of the protocol analysis and the experimental results, see [31,33].

2.4. Diagnosis and learner behaviour

Assessment of learner behaviour is often referred to as cognitive diagnosis and viewed as similar to student modelling [58]. In this section we review the most typical approaches in this area (Section 2.4.1), and argue that there is an important difference between the diagnosis of learner behaviour and the maintenance of a learner model (Section 2.4.2). We then discuss model-based techniques for diagnosing device behaviour (Section 2.4.3), and

present the hypothesis that the realisation of behaviour diagnosis in educational systems may benefit from using these techniques (Section 2.4.4).

2.4.1. Cognitive diagnosis and learner modelling

Extensive research exists on learner modelling and cognitive diagnosis in Intelligent Tutoring Systems (ITS) [24,35,45,63,75]. Two dominant approaches emerge from this literature: *overlay* and *perturbation*. Overlay models keep track of the knowledge that a learner has acquired by recording for each entity in the subject matter model whether it is known by the learner or not. The underlying idea is that the subject matter model represents the expert behaviour, and that all differences between the learner's behaviour and that of the expert model can be explained in terms of the learner lacking certain knowledge. An overlay model was used in many early educational systems [3,21–23]. The overlay model works well in situations where the knowledge represented in the subject matter model can be transferred directly to the learner. This is true for teaching factual knowledge such as geography or English vocabulary, and for a limited set of procedural domains (e.g., training fixed safety-critical procedures in nuclear power plants). The usability of overlay models significantly decreases when the subject matter requires the acquisition of complex problem solving skills and the learner is allowed an increasing freedom in how to approach the teaching material.

The second approach views the learner's knowledge not as a subset of an expert's knowledge, but as intersecting with it: part of the learner's knowledge will be 'correct' (i.e., identical to the subject matter model), and part of it will be different. This 'different' knowledge is usually referred to as *misconceptions* or *bugs*. Hence, the learner model consists of an overlay of the subject matter model, possibly extended with a number of buggy facts or procedures. One of the key ideas behind the perturbation approach is that these extensions *explain* the learner's erroneous problem solving behaviour. Two approaches exist. One approach uses explicit representations of the bugs stored in a *bug catalogue* [16,17,19,25,44,50]. The advantage of a bug catalogue is that, once it is available and complete, the diagnostic process is reduced to 'comparing' erroneous facts and procedures (*mal-rules*, [65]) with the learner's problem solving behaviour. However, the usability of bug catalogues is hampered by the fact that their construction is extremely laborious. Furthermore, bug catalogues are domain-specific and hence not reusable. Also it is impossible to ensure that a bug catalogue is complete. The alternative approach tries to overcome these difficulties by *reconstructing* or *generating* bugs dynamically. To this end, generative theories of bugs were developed [6,18,72]. Generative systems are based on cognitive theories on how bugs come into being. For instance, REPAIR Theory [18] centres around the idea of repair of impasses in *impasse-driven problem solving*; an impasse occurs when no further relevant knowledge is available or when conflicting knowledge is applied, resulting in the learner being unable to proceed. Learners will now search for repairs to fill this knowledge gap. Applying the repair to an impasse may lead to a bug. Although no hand-crafted bug catalogues are needed, a disadvantage of generative theories is that domain-specific filters have to be used to avoid the generation of implausible bugs. Moreover, the construction of generative theories has proven to be difficult, mainly as a result of lacking operational theories on how people acquire knowledge.

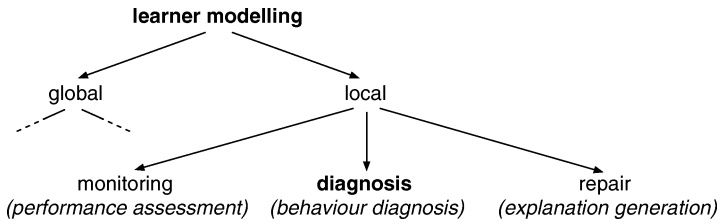

Fig. 4. A functional decomposition of performance-based learner modelling.

2.4.2. Behaviour, bugs and misconceptions

In contrast to most existing approaches we explicitly distinguish between behaviour diagnosis and learner model construction. The different roles are illustrated in Fig. 4 by means of a *functional decomposition* of the learner modelling task [9,13]. The global part of learner modelling refers to the actual construction of the learner model. This model typically contains data of very different types such as the name of the learner or the number of errors made in the last exercise, and may persist over several educational sessions. In addition, this model may include information about what the learner is believed (not) to know, including certain misconceptions. However, as mentioned before, a full realisation of the latter requires an operational theory about human learning. The research presented in this article is concerned with the local part of learner modelling. This part, which in principle is independent from the global part, can be divided into three major functions: monitoring, diagnosis, and repair [14,80]. As indicated in the figure, these functions correspond to three educational functions in terms of our architecture (Fig. 2): performance assessment, behaviour diagnosis and explanation generation. The function of diagnosis is limited to finding a consistent *explanation* of some behavioural deviation and is not concerned with identifying the deviation itself. The latter is part of the monitoring function, i.e., performance assessment [15]. The repair function comprises the activities that can be undertaken to remediate the deviation in the learner's problem solving behaviour. Notice, that the learner model may be used as an additional input for each of these functions (see also Fig. 2), but this is not a prerequisite. In the case of diagnosis such extra input could be used as a *focus* [27].

In existing approaches, the *purpose* of diagnosis is often said to *explain* or indicate the *cause* of errors made by the learner. Such causes are referred to by various terms, such as *bugs* or *misconceptions*. However, there is an important distinction between the two, as observed in [35]. The difference can be clarified by placing bugs on the *behavioural level*, and misconceptions on the *conceptual level*. Hence, misconceptions are higher-level deviating or erroneous conceptions about the domain, such as "heat and temperature are the same thing". Bugs are behavioural errors, and can as such be manifestations of misconceptions. For instance, the derivation "the heat is constant, and therefore the temperature as well" can be a bug. To complete the terminology, *errors* are deviations in the output of the learner. The answer "the temperature is constant" is an example of a possible error. Summarising, errors are manifestations of bugs, which in turn may be caused by misconceptions.

We focus on detecting bugs rather than misconceptions. This means that we restrict ourselves to diagnosing the problem solving of the learner at the behavioural level, rather than trying to find explanations of the learner's (mis)conceptions about the domain. Each problem solving process can be viewed as the application of a set of individual inference or reasoning steps (cf. [78]). Hence, we define a diagnosis as follows:

> A diagnosis is a (minimal) set of reasoning steps that cannot have been performed correctly by the learner given the observations

A diagnosis explains the behaviour of the learner in terms of *how* this behaviour deviates from the norm behaviour, rather than *why* this happened. A major advantage of this approach is that it can be based solely on a model of the reasoning steps that are required for the problem solving process; no knowledge is required about the specific misconceptions that learners may have about the domain. Note that by concentrating on individual reasoning steps, we do not prescribe the order in which reasoning steps are applied (i.e., the problem solving strategy). For instance, it does not matter whether a learner derives the boiling of a liquid by first recognizing that the temperature of a liquid is below it's boiling point, and then that it is increasing, or the other way around.

2.4.3. Model-based diagnosis

In model-based diagnosis [47], the basic idea is that faults appear as a discrepancy between the behaviour of a device, and the prediction of that behaviour made by using a model of the device. An important consequence is that all erroneous behaviour is defined in terms of its deviation from the correct behaviour (i.e., the behaviour as represented in the model). Therefore, model-based diagnosis does not necessarily incorporate fault models, and thus does not depend on the completeness of these fault models. The notion of *consistency* is mostly used to define diagnoses. Initially, all components are assumed to be operating correctly. In the case of a discrepancy these assumptions are reconsidered. The diagnosis, defined as a set of model components for which the behaviour is unspecified, is the set of components for which the correctly operating assumption has to be retracted in order for the system behaviour as a whole to be consistent with the observations.

One of the most influential diagnostic frameworks that has been developed is the *General Diagnostic Engine* [30]. This approach is meant to provide a general framework for model-based diagnosis, being capable of handling single as well as multiple faults. The diagnostic process is completely separated from the mechanism for the prediction of behaviour. The device model used in GDE consists of the device components, their connections, and the behavioural constraints for each component. Furthermore, with each component an initial assumption is associated stating that the component is functioning correctly. The diagnostic procedure consist of three main steps: conflict recognition, candidate generation, and candidate discrimination. A *conflict* is a set of component assumptions that conflicts with the observations. That is, if each component in a conflict is assumed to behave correctly, the predicted behaviour of the device is inconsistent with the observations. Conflict generation is guided by observations: after each measurement, the new observation is employed to find new conflicts. Only subset-minimal conflicts have to be located, because each superset of a conflict is automatically a conflict as well. The search for minimal conflicts is done by

selecting so-called *environments*, sets of assumptions which are all assumed to be true. Each environment is tested for consistency with the observations. If an inconsistency is encountered, then the current environment is a conflict. A *candidate* in the GDE paradigm is a hypothesis about the difference between the actual artifact and the correct model. A candidate defines a set of components that are all considered to be malfunctioning. Like in the case of the conflicts, only minimal candidates are generated. Each candidate has to account for all the known conflicts, and therefore should have at least one component in common with each conflict. The ultimate goal of diagnosis is to find one minimal candidate that accounts for all conflicts. The set of candidates is pruned by selecting the most discriminating probe based on 'minimal entropy', a concept known from the field of decision and information theory.

A major problem of model-based diagnosis is its computational complexity: the diagnostic problem as it is defined in GDE and other model-based diagnostic systems is NP-hard [20]. In model-based reasoning, this problem is alleviated by improving the algorithms of the diagnostic engine (e.g., [27]) or by introducing more advanced knowledge representations, such as *hierarchies* (e.g., [43,46,57]). In the case of the latter, the top level of the model provides only an abstract view on the system modelled. When a component of the top-level model is identified as relevant to be explored in more detail, that component is decomposed into a set of lower-level components.

2.4.4. Model-based diagnosis of learner behaviour

Following ideas presented in [62], we propose a model-based approach to diagnosis of learner behaviour. In ITS research, model-based diagnosis has not been applied often. The work of Huang et al. [49] that uses Reiter's HS-Tree algorithm [60] comes very close, and also the logic-programming, deductive approach presented in [4,48] is a form of model-based diagnosis, but references to existing research on the subject in the field of artificial intelligence are hardly made within these publications.

Recall that in model-based diagnosis, a diagnosis is defined in terms of defective model components. In our approach, each component represents an instantiated reasoning step in the problem solving process. These components are used to define the diagnostic model as a *reasoning trace* incorporating all individual reasoning steps that must be mastered by the learner to derive the correct solution. In accordance with this model definition, a diagnosis is a set of reasoning steps that the learner cannot have applied correctly given the observations.

In this respect it is interesting to note that in the Wizard-of-Oz experiment (see Section 2.3 and [33]) teachers never came forward with remarks concerning why the learner made an error. Instead they were mainly concerned with what the learner had done wrong in relation to how the problem should have been solved, i.e., the norm model. This supports the hypothesis that automating this assessment activity for simulation-based learning environments is an important breakthrough in realising a knowledgeable communication between such environments and learners.

Although the advantages of using model-based diagnosis in education are considerable in terms of generality, reusability, and transferability, a major bottleneck is the fact that the behaviour models of the 'correct problem solving behaviour' are not readily available [62]. As such, the costs of building domain-specific bug catalogues, or developing a generative

theory of bugs, may seem to be replaced by the cost of building domain-specific diagnostic models. To solve this problem we propose to generate the diagnostic domain models automatically from the output of a qualitative simulator, in which case the advantage is evident. This requires that the output of the simulator can be transformed to meet the rather strong requirements for model-based diagnosis (Section 3). In addition, the computational complexity of model-based diagnosis needs to be addressed (Section 4).

3. The base model

The first problem that should be tackled in order to apply model-based techniques to reasoning knowledge is the mapping of this knowledge onto the model-based paradigm. This section discusses the definition of problem solving knowledge in terms of component-connection models. The resulting models are referred to as the *base models*.

3.1. Representation of the base model

For a specific prediction of behaviour, the base model represents the set of all reasoning steps or *inferences* that are required for this prediction. An inference can be defined by an input, an output, and some (generic) support knowledge to derive the output from the input (e.g., [68,79]). For instance, an *influence* from quantity A to B serves as support knowledge for deriving the derivative of B from the value of A. In the base model, each individual reasoning step is represented as a *component*. More precisely, each *application* of an inference is represented as a component in the model. A base model component has a non-empty set of input ports, a possibly empty set of generic support knowledge ports, and one output port.[3] Each component port is connected to exactly one measure point, but one measure point can be connected to more than one component port. If a measure point is only connected to input (or only output) ports, the point is a model input (or output). The *data flow* through these connections is formed by instantiated *expressions* (inequalities, causal dependencies, or quantity spaces).

A base model fragment is shown in Fig. 5, where the behaviour of a single emptying container is considered. The model represents the derivation of two terminations: 'volume is going to zero' ($V > 0 \rightarrow V = 0$) and 'level is going to zero' ($L > 0 \rightarrow L = 0$). For the leftmost component in Fig. 5, called quantity correspondence, the support knowledge expression is $dir\_corr(V, L)$,[4] the input expression is $V > 0$, and the output expression is $L > 0$. The behaviour of a component is defined by a procedure or *behaviour rule* that calculates the output from the dynamic and static (support) inputs:

IF $V > 0 \,\&\, dir\_corr(V, L)$ THEN $L > 0$.

Reasoning from the value of the volume V, we can subsequently derive the value of the level L, the pressure P and the flow rate Fl. The derivative of the volume δV is

[3] Note that the support knowledge is not the same as the behaviour rule of the component: given an influence relation $pos\_infl(A, B)$ as support knowledge, a behaviour rule can be defined that *uses* this relation in calculating δB from δA.

[4] *dir_corr* refers to 'directed quantity space correspondence' (Section 2.1).

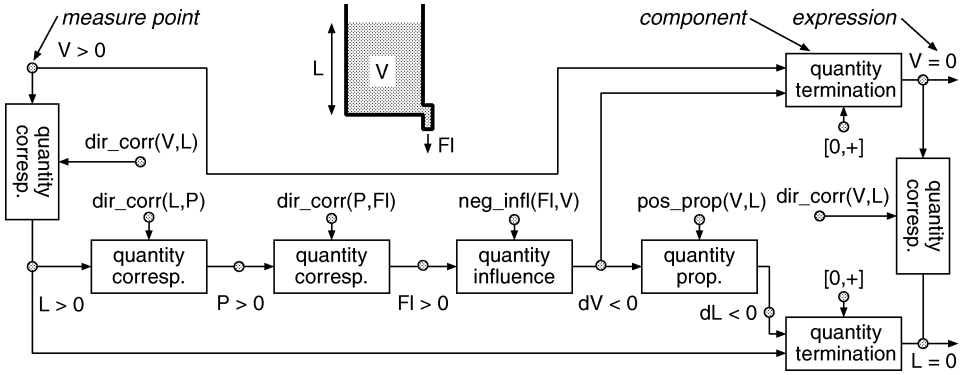

Fig. 5. The base model representation.

derived from Fl by a quantity influence component. The negative value for δV, together with the given model input $V > 0$ and the quantity space $[0, +]$, allows for the derivation of $V = 0$ in the next state by a quantity termination component. Similarly, the termination $L > 0 \rightarrow L = 0$ can be derived by calculating the derivative of the level (δL) from δV, and applying an equivalent quantity termination.

Based on previous research (see Section 2.3), ten main component types have been defined for the construction of base models. These components deal with correspondences, proportionalities, influences, terminations, and determinations. The latter inferences are

Table 1
Basic inference component types

Component	Example input(s)	Example support knowledge	Example output
Quant. correspondence	$A > 0$	$dir\_corr(A, B)$	$B > 0$
Quant. proportionality	$\delta A > 0$	$pos\_prop(A, B)$	$\delta B > 0$
Quant. influence	$A > 0$	$pos\_infl(A, B)$	$\delta B > 0$
Quant. termination	$A > 0, \delta A < 0$	[0,+]	$A = 0$
Ineq. correspondence	$A_1 > A_2$	$dir\_corr(A, B)$	$B_1 > B_2$
Ineq. proportionality	$\delta A_1 > \delta A_2$	$pos\_prop(A, B)$	$\delta B_1 > \delta B_2$
Ineq. influence	$A_1 > A_2$	$pos\_infl(A, B)$	$\delta B_1 > \delta B_2$
Ineq. termination	$A > B, \delta A < \delta B$	—	$A = B$
	$A > B, \delta A < 0, \delta B > 0$	—	$A = B$
Value determination	$A = B$	$C = A - B$	$C = 0$
	$A > 0, B > 0$	$C = A + B$	$C > 0$
Deriv. determination	$A > B$	$\delta C = A - B$	$\delta C > 0$
	$A > 0, B = 0$	$\delta C = A - B$	$\delta C > 0$

used to define a value or derivative in terms of the sum of or difference between other values or derivatives; e.g., the position of a balance in Fig. 3 can be determined from the difference between the weight at the left and right side. Different component versions exist for manipulating individual quantity values and derivatives, and for (in)equalities (Table 1).

To convey the basic idea behind the component definitions, we discuss one component type in more detail: the *quantity influence*. For the remaining component definitions, the reader is referred to Appendix A.

Influences between quantities induce change: given a value for A, and an influence of A on B, derive a derivative for B. To this end, a quantity influence component has one quantity value as input, one influence as support knowledge input, and a quantity derivative as its output. For these ports, the behaviour rules that define the "forward" behaviour of the inference component is shown below. The syntax of the behaviour rules is as follows: 'In' denotes the input(s) of the component, 'Sup' denotes the support knowledge, and 'Out' denotes the output expression of the component. Expressions are printed in square brackets. When different possibilities for one expression are given, such as in $[\delta A = -/0/+]$, the positional equivalent expression should be chosen in other parts of the rule.

Forward behaviour rules: In & Sup $\rightarrow$ Out

$$\text{IF In} = [A >/=/< 0] \,\&\, \text{Sup} = [\text{pos\_infl}(A, B)] \text{ THEN Out} = [\delta B = +/0/-]$$
$$\text{IF In} = [A >/=/< 0] \,\&\, \text{Sup} = [\text{neg\_infl}(A, B)] \text{ THEN Out} = [\delta B = -/0/+]$$

Backward behaviour rules:

$$\text{Out} \,\&\, \text{Sup} \rightarrow \text{In}$$
$$\text{In} \neq 0 \,\&\, \text{Out} \neq 0 \rightarrow \text{Sup}$$

Backward behaviour rules can be defined analogously[5] except for one case: when the derivatives in the input and output expression are zero (therefore: $\text{In} \neq 0 \,\&\, \text{Out} \neq 0$), then the influence relation at the support knowledge input (Sup) cannot be uniquely determined. For example, from an input $A = 0$ and an output $\delta B = 0$, we cannot determine whether the support knowledge is *pos_infl*(A, B) or *neg_infl*(A, B).

3.2. Modelling competing inferences

In qualitative prediction, causal dependencies and state changes are not always independent, and can interfere with each other. For instance, two quantities may have opposite influences on the derivative of another quantity, or a change in a quantity value may not occur (yet) because another change takes place first. We refer to such inferences as *competing*. The representation of competing inferences in the base model is problematic, because the model-based reasoning representation employed in the base model requires the components to obey the *no-function-in-structure* principle [28]. This means that the behaviour of each component should be defined independent of its context, i.e., independent of the behaviour of other components. Below, we discuss how respectively competing dependencies and competing terminations are represented in the base model.

[5] Substitute the Out, Sup and In as specified in the forward rule to reconstruct the backward rules.

3.2.1. Competing dependencies

Consider the container system shown in Fig. 6, where water flows in from the tap, and at the same time water flows out from the outlet at the bottom. Given a ratio of the inflow Fl_{in} and the outflow Fl_{out}, say $Fl_{in} < Fl_{out}$, the derivative $\delta V < 0$ can be derived for the volume V. When no explicit constraint is modelled between such competing influences the situation is ambiguous. Qualitative simulators typically generated all possible states of behaviour in such situations (here, one state for each possible value of δV) but they usually do not specify the different ratios between the competing influences that are true for each of the behaviours (here: $Fl_{in} < Fl_{out} \rightarrow \delta V < 0$, $Fl_{in} = Fl_{out} \rightarrow \delta V = 0$ and $Fl_{in} > Fl_{out} \rightarrow \delta V > 0$). Hence, competitive causal effects are always governed by constraints on the ratio of the inputs of the components modelling the causal effects, although these constraints are not always explicitly available in the simulator's output.

To obey the no-function-in-structure principle, the behaviour of each component must be determined solely by its inputs, without making reference to (the behaviour of) other components. For competitive dependencies, one solution would be to define separate *combination* components. In the case of the above example, this 'combination' component would have $Fl_{in} > 0$ and $Fl_{out} > 0$ as inputs, the influences *pos_infl*$(Fl_{in},\ V)$ and *neg_infl*$(Fl_{out},\ V)$ as well as the constraint $Fl_{in} < Fl_{out}$ as support knowledge, and deliver $\delta V < 0$ as an output. This approach has several drawbacks however. Firstly, the above 'add' definition only works for *two* competing dependencies. If three or more depedencies are competing, the number of inputs and supports will increase. Hence, the structural definition of the 'combination' component is variable with respect to the number of competing depedencies. Secondly, the complexity of the component is also disadvantageous from a computational point of view. The rules that govern the component's behaviour are very complex, and backward propagation of values is impossible for most cases. Finally, a problem related to the previous one is that the semantics of the 'combination' component would become rather complex. Even for the simplest case with two competing dependencies, the number of ports is relatively large, and advanced techniques would be needed to explain or ask questions about the component in an educational context.

Other solutions, for instance using explicit 'add' components for adding dependencies two at a time, yield similar problems: both the computational properties and the semantics are problematic [31]. Given the educational purpose of the representation, we propose a simpler solution. We define extra types of quantity proportionality and quantity influence components, called *submissive*. As can be seen in Fig. 7, this allows for explicit

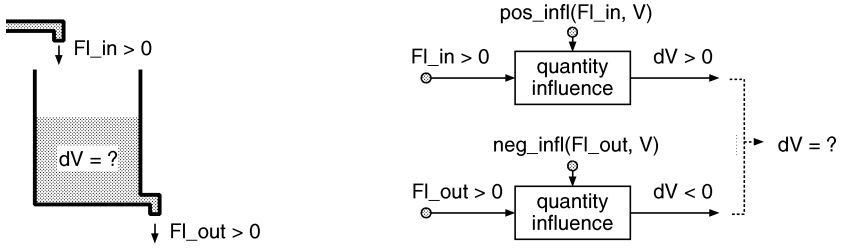

Fig. 6. Representing competitive dependencies.

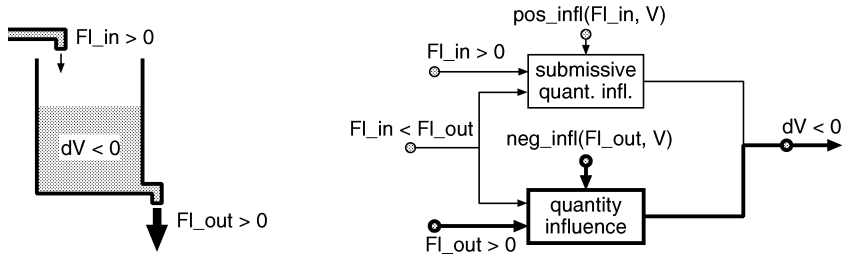

Fig. 7. Submissive components.

representation of 'competing' influences. These submissive influence components have an extra input, representing the constraint that was needed to resolve the conflict. The same extra input is also input to the corresponding non-submissive influence or proportionality component (the lower component in Fig. 7). When no explicit constraint is available in the simulator's output, it is generated automatically. [6]

By modelling submissive causal relations this way, the educational system has available all information needed for explaining the competitive effects. Although no backward reasoning is possible through submissive components, because of its ambiguity, a major advantage over other solutions is the clarity of this solution.

3.2.2. Competing terminations

Similar to what happens when proportionalities and/or influences cannot be applied unambiguously, not all state terminations that are possible will occur. As an example, consider the balance system in Fig. 8(a). In this situation, the correct transition to the next state contains, among others, the termination from $Pos = 0$ to $Pos < 0$: the balance position changes from equilibrium to 'left hand side down' (Fig. 8(b), upper one). The termination of $V_l > 0$ to $V_l = 0$ (the left container becomes empty) is *not* applied, although the conditions are met (Fig. 8(b), lower one). The order in which terminations happen is governed by *precedence rules* (Section 2.1), in this case the *ε-ordering rule* [28]: the result of the former termination is immediate, whereas the latter is not. More precisely, a change from a point ($Pos = 0$) to an interval ($Pos < 0$) is instantaneous, whereas a change from an interval ($V > 0$) to a point ($V = 0$) is not. Therefore the former termination will have precedence over the latter one. Hence, V_l *remains* positive in the next state: the termination component is *submissive* here.

This idea of using submissive termination components to 'transport' non-changing values to the next state shows similarity with the use of *continuity rules* in GARP (see Section 2.1). The main difference is that continuity rules do not distinguish between terminations that do not occur because they are 'overruled', and quantities that have an explicit reason not to change, namely a zero derivative. We use submissive terminations only for terminations that are 'overruled', and define continuity components to account for transporting quantities with a zero derivative to the next state. A *continuity component* is

[6] Because the base model is generated on the basis of an existing behaviour simulation, we can always generate this constraint from the available knowledge in the simulator's output (see Section 3.3).

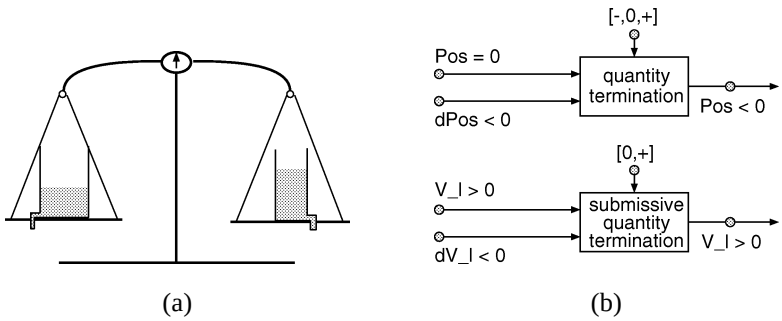

Fig. 8. Submissive terminations.

defined analogously to a termination component, except that derivative inputs are required to be zero.

3.3. Base model generation

The base models are generated automatically from GARP's output. This simulation model contains all correct *facts* that we want the learner to know or derive, and as such defines the base for the subject matter model. A post-processor was developed that takes these facts, generated by the simulator, and creates a base model measure point for each of them. The facts define the expressions at the created measure points. Subsequently, all possible valid *reasoning steps* are added to the base model. This is done by matching the behaviour rules of each component type (see Table 1) to the expressions, and adding each component that represents a valid inference step in the model. This way, the output of GARP is used to build an explicit model of the reasoning knowledge, while the adequate grain size of the reasoning steps is guarded by the component definition (cf. Section 2.3). An example base model is discussed in Section 4.4.

4. Model aggregation

The base model represents the reasoning knowledge we want to interact about with the learner, modelled at the most detailed level still relevant for education. However, a complete prediction for the balance system (Fig. 3), consisting of six behavioural states, yields a base model of 665 components and 612 points. This size makes the model hardly suitable for model-based diagnosis in a run-time learning environment. Although in theory such a number is not prohibitive for real-time diagnosis [27], the base model has a number of specific characteristics that are disadvantageous with respect to model-based diagnosis. Firstly, the connectivity between components is relatively low, as compared to digital circuits. Secondly, the number of observations, and particularly observed outputs, is low. Finally, because the calculi underlying qualitative reasoning are relatively weak, the definition of behaviour rules for backward propagation is not possible for all component types. The combination of these characteristics together with the large number of components provides a 'worst case scenario' for model-based diagnosis. That is,

the situation is highly underconstraint leading to large numbers of diagnosis being found. Although none of the above-mentioned characteristics of the model violates an explicit requirement for model-based diagnosis, no 'typical' models (read: electronic circuits) have these characteristics.

To make things tractable, we apply hierarchical modelling techniques. Hierarchical structuring requires two issues to be addressed. Firstly, the hierarchical structure must be generated automatically from the the contents of the base models. In artifacts, the hierarchical structure is often evident from the physical and/or functional structure of the device. Devices are usually *designed* hierarchically for reasons of component reuse and maintainability, and hence the hierarchical diagnostic models can be derived from the blue prints [43]. Such a natural decomposition is not available for the kind of norm models used in our approach. The inference components that are part of the base model do not necessarily map onto components in the 'real' system: they do not have any serviceable physical counterpart, and no blue prints are available. Despite this problem we have to find methods for automatic aggregation.

Secondly, hierachical modelling has not only computational advantages, but from an educational point of view it is also a necessity. A set of 665 reasoning steps, as required for predicting the behaviour of the balance system, is large and therefore difficult to communicate without any further means of distinguishing the crucial reasoning steps from the less essential ones. Given this additional purpose of the hierarchical models, the hierarchical structure should preserve the cognitive nature of the base model as much as possible. The measure points that will be available on each of the aggregation levels play a major role. For adequate diagnoses, there are preferably few measure points on higher levels, which should also be easily measurable. In an artifact, the costs of a measurement are determined by factors like physical reachability and the cost of the measuring device needed. Measurements in the models that we use have different characteristics, and hence different 'costs': in an educational context, probes are taken by asking questions. The costs are determined by the discourse context (a question should fit in with the current topic), but also by the knowledge level of the learner (the learner should be able to understand and answer the question).

The two aspects discussed above are operationalised in the following main principles for hierarchical model generation. The first principle, referred to as *hiding non-essential details*, results in some components and points of the model to be discarded at a higher level, because they do not belong to the main derivation traces in the prediction at hand. The second principle, referred to as *chunking*, amounts to replacing a sequence of components by one abstract inference component, thus combining a chain of reasoning steps into one.[7] In addition to these principles, we exploit the existing partition in the simulator's output into behavioural states and transitions between these states. That is, at the most abstract level, we *group* the different inference components according to the behavioural state or transition to which they belong. Below, we present the aggregation principles. The algorithms that implement the aggregation can be found in Appendix B.

[7] The names of these principles suggest relations to learning and teaching theories, and although such relations exist, there are important differences (see also Section 7).

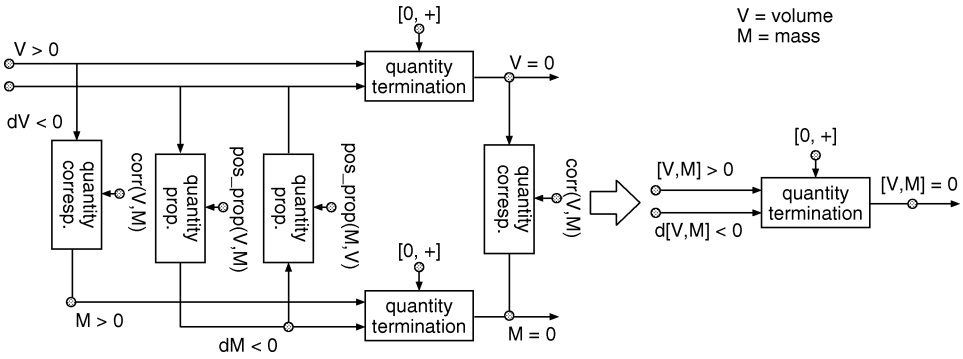

Fig. 9. Hiding of non-essential details: fully-corresponding quantities.

4.1. Hiding non-essential details

The first notion of non-essentialness is concerned with quantities that play similar roles in the problem solving process. Such quantities are called *fully-corresponding*. Pairs of fully-corresponding quantities are abstracted into one new quantity, and hence all inference components that manipulate them are integrated. For example, the quantities mass and weight can be considered equivalent when notions such as gravitational force are abstracted from. An example of fully-corresponding quantities is presented in Fig. 9, where for an emptying container the quantities mass and volume of the liquid are combined. Note that the fact that two quantities are fully corresponding depends on the modelling assumptions made: in the above example, the full correspondence between the volume and mass of a liquid depends on the assumption that this liquid is a homogeneous mass.

Fully-Corresponding Quantities. For two quantities A and B with dependencies $corr(A, B)$, $pos\_prop(A, B)$, and $pos\_prop(B, A)$, a combined value $value([A, B])$ and derivative $derivative([A, B])$ are defined.[8]

All data inputs and outputs concerning either A or B are replaced by $[A, B]$, and all duplicate derivations are removed.

The conditions for quantities to be considered fully corresponding incorporate two proportionalities and an undirected correspondence between those quantities. We apply hiding as a mechanism for abstracting from what one could call *qualitative synonyms*: those quantities that are behaving *identically* in the context of this specific simulation. Weakening the notion of full correspondence, by for example only requiring one proportionality, may yield problems with directions of causality: fully-corresponding quantities can then no longer be guaranteed to behave identically. In simulation practice, qualitative models usually do not contain opposite proportionality relations. We define the notion of full correspondence for educational reasons, such that information about the relation between

[8] In the figures, fully-corresponding quantities should not be confused with quantity spaces, which are also represented in square brackets.

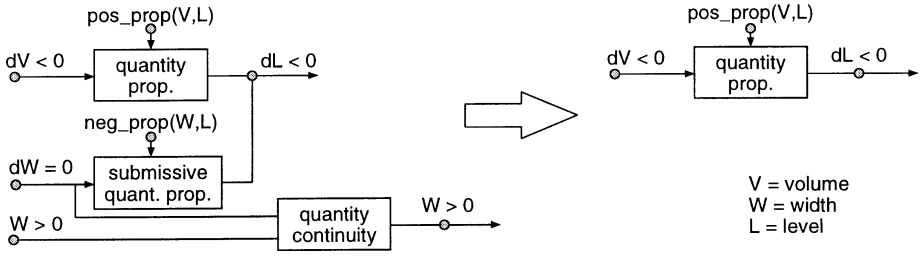

Fig. 10. Hiding of non-essential details: removing submissive components.

fully-corresponding quantities can be used explicitly in the interaction with a learner. Accordingly, full correspondence between different quantities is not something that is 'discovered' by the aggregation algorithm in arbitrary models, but it should be modelled explicitly as such by the constructor (i.e., in terms of the model fragments that GARP uses for generating a simulation model).

The second form of hiding consists of leaving out submissive causal relation components, submissive termination components, and continuity components. Submissive components represent inferences that are usually not reported by learners or tested by teachers unless there is a local misunderstanding [33]. Therefore, we can hide submissive components. An example for the emptying container is provided in Fig. 10: the quantity width, that does not play an active role in the prediction, is abstracted from.

The third and last form of hiding consists of the removal of inactive paths in the model. Because of earlier hiding steps, some inference paths may have become "dead ends", and hence their derivation is no longer essential to the prediction. Removal of inactive paths in the aggregation is also done in the context of chunking (see below), and its definition is context dependent.

4.2. Chunking

The next step in model abstraction is chunking: when a number of steps in the reasoning process are directly related, and the data derived by the intermediate steps are not relevant to other parts of the reasoning process, then these steps can be taken together. We employ two chunking principles, differing in the types of components involved: *transitive chunking* and *key component chunking*.

Transitive chunking amounts to chunking components of the same type: both correspondence and proportionality relations are transitive, and hence subsequent components of such types can be chunked. The notation below uses the terms input(C) and output(C) to refer to the input and the output *points* a component C is connected to.

Transitive Chunking. Two components C_1 and C_2 of equal type quantity correspondence, inequality correspondence, quantity proportionality, or inequality proportionality, with output(C_1) = input(C_2) are combined into a new transitive component C_n with input(C_n) = input(C_1) and output(C_n) = output(C_2).

output(C_1) should not be connected to any other specification component than C_2 (the *non-branching condition*).

In general, intermediate data can only be omitted when it is not input to some other inference component, i.e., when the sequence of inferences is non-branching. In our model, however, this non-branching condition is weakened to allow one specific type of branching, namely to termination components. In fact, we want intermediate quantities to stay intermediate when they are part of a group of quantities that all change values at a certain state change. To explain this, consider Fig. 11, where for the emptying container the derivative of the flow rate is derived to be negative (i.e., the flow is decreasing). In this derivation, there are branches to termination components from each of the intermediate points (which will derive $V = 0$, $L = 0$, $P = 0$, and $Fl = 0$ respectively). Following the definition presented above, chunking is not possible here. When replacing, say, the two upper-left quantity correspondence components by introducing a component deriving $P > 0$ from $V > 0$, one input of a termination is lost: $L > 0$ is no longer explicitly derived. However, the other input of that termination component ($\delta L < 0$) can also be removed by chunking the two lower-left quantity proportionality components. In this case, we can remove the termination component from the abstract model, under the additional requirement that (in the next state) the output of the termination can also be derived by other means. In the above example, the model contains a quantity correspondence component that derives $L = 0$ from $V = 0$ in the next state. Fig. 12 shows an example of how chunking is implemented by means of both transitive correspondences and transitive proportionalities. The model depicted is the 'chunked' version of Fig. 11. The quantities level and pressure are abstracted from in the higher-level model, enabling inferences like "there is [a positive volume of] water in the container, so it will flow out".

The next step in chunking is *key component chunking*, which is based on the idea that some component types are more central to the behaviour of a system than others. We consider influences and value/derivative determinations to be the *key components* in the base model. The rationale behind this definition is as follows. Influences represent the existence of a *process*, which is considered an important concept in system behaviour [38]. Value and derivative determinations represent the definition of a new quantity in terms

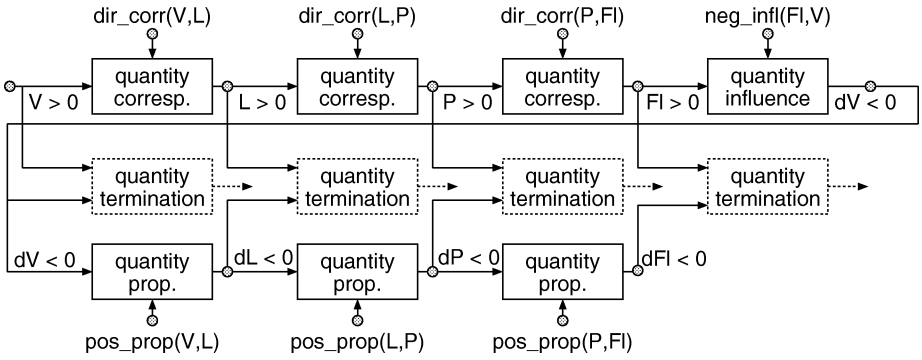

Fig. 11. Sample derivation before chunking.

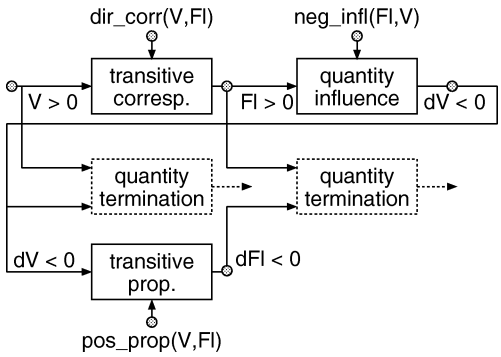

Fig. 12. Sample derivation after transitive chunking.

of others. In general, quantities that represent a difference or sum are considered to play a more important role in the prediction than quantities whose value and derivative are derived by propagation through correspondences and proportionalities. Note that both types of key components are often (but not always) related: a typical example of a quantity that is defined by a value determination is a flow rate (i.e., $Fl = T_s - T_g$), which in turn is input to a quantity influence (i.e., $pos\_infl(Fl, H_g)$).

Key components are chunked in two phases: first their input is chunked with the preceding component(s) (*predecessor chunking*), next the result is combined with the succeeding component(s) (*successor chunking*). The definition for chunking a quantity influence with its predecessor is given below; for an exact definition of other possible chunks, see Appendix B.

Combined Quantity Influence. A component C_{infl} of type quantity influence can be combined with a component C_{corr} of type (transitive) quantity correspondence if output(C_{corr}) = input(C_{infl}), and output(C_{corr}) is not connected to any other specification component than C_{infl}.

A new component C_n of type combined quantity influence 1 is defined with input(C_n) = input(C_{corr}) and output(C_n) = output(C_{infl}).

An example is given in Fig. 13, where the model from Fig. 12 is further abstracted by chunking the transitive correspondence component with the quantity influence component. The transitive proportionality component in Fig. 12 can be removed because its only

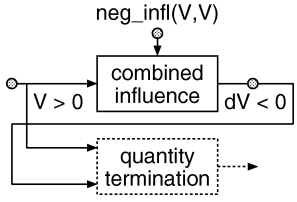

Fig. 13. Sample derivation after predecessor chunking.

purpose is to derive an input for the termination component. This termination component is removed by the chunking algorithm: the output $Fl > 0$ of the termination component in the next state is also abstracted from by the chunking algorithm.

4.3. Grouping

As a third and last step, we generate abstracted models by grouping. Grouping is inspired by the educationally relevant subtasks that can be distinguished in prediction of behaviour: *specification* of individual qualitative states, and *transitions* between these states. Grouping should be conceived as an additional top-level structuring of the subject matter rather than as an operational hierarchical layer: no behaviour rules are defined for specification and transition components. In the description below, the super type transition refers to all (submissive) termination and continuity component types, and specification covers the remaining component types.

Grouping State Specifications. All components C_i of type specification that belong to the same state are combined.

A new component C_n is defined with

$$\text{expression\_set}(\text{input}(C_n)) = \bigcup \text{expression}(\text{input}(C_i)) \quad \text{and}$$

$$\text{expression\_set}(\text{output}(C_n)) = \bigcup \text{expression}(\text{output}(C_i)).$$

Grouping State Transitions. All components C_i of type termination that belong to the same transition are combined.

A new component C_n is defined with

$$\text{expression\_set}(\text{input}(C_n)) = \bigcup \text{expression}(\text{input}(C_i)) \quad \text{and}$$

$$\text{expression\_set}(\text{output}(C_n)) = \bigcup \text{expression}(\text{output}(C_i)).$$

Instead of single expressions, the data flow between the components consists of *sets* of expressions; the notation expression(P) is used to refer to the expression carried by a point P.

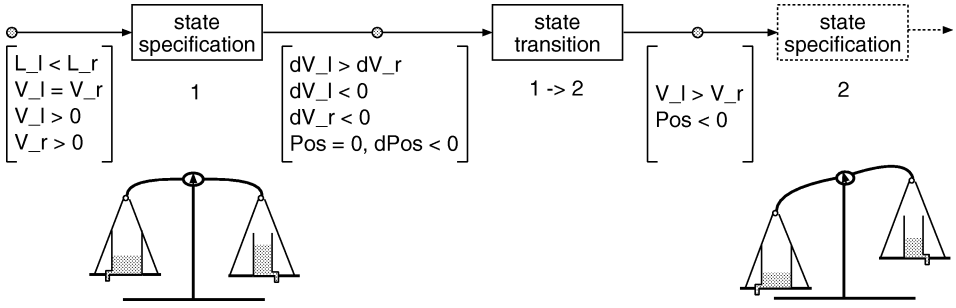

Fig. 14. Grouping of states and transitions.

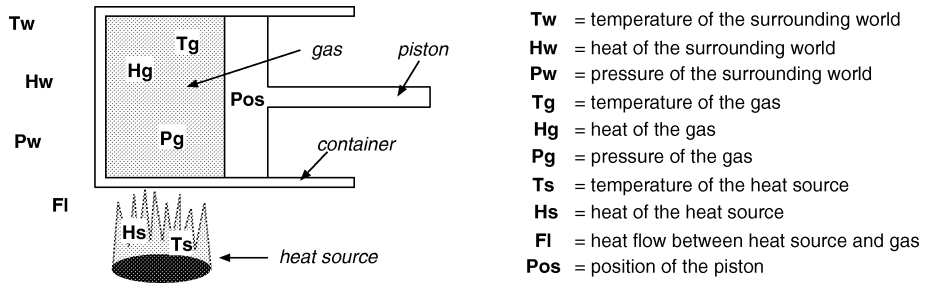

Fig. 15. The piston system.

Because grouping is applied as the last aggregation step, the expressions flowing between the different specification and transition components are likely to be small in number; many expressions in the base model are already abstracted from in the other aggregation steps. To exemplify this, consider the high-level model for the balance domain depicted in Fig. 14. Because we abstracted from various intermediate quantities at lower levels, such as pressure and flow rate, the state description that forms the output of the first state specification is relatively small. If we apply grouping directly to the base model, then this state specification output contains 13 instead of 5 elements.

4.4. An example model

As an example of the modelling and structuring principles discussed above, consider the piston system in Fig. 15. The piston system consists of a movable piston in a container.

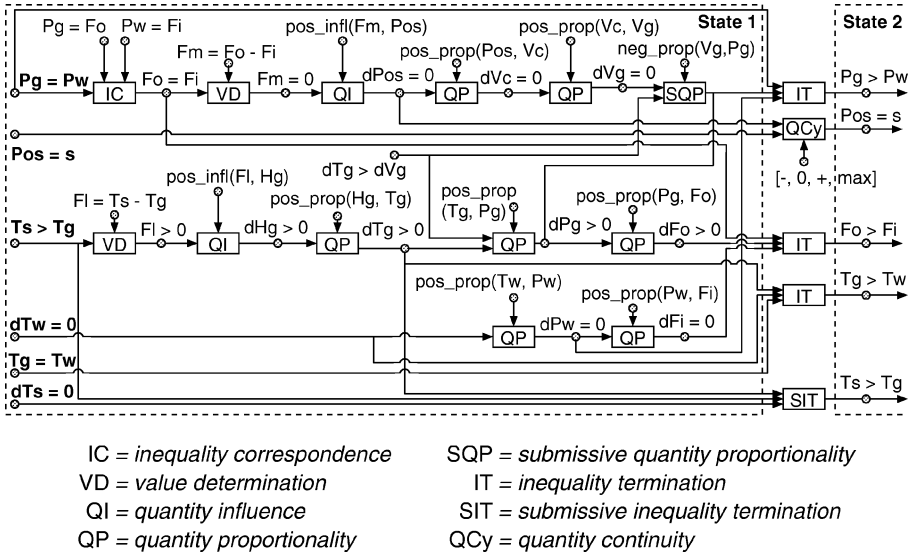

Fig. 16. Part of the base model for the piston.

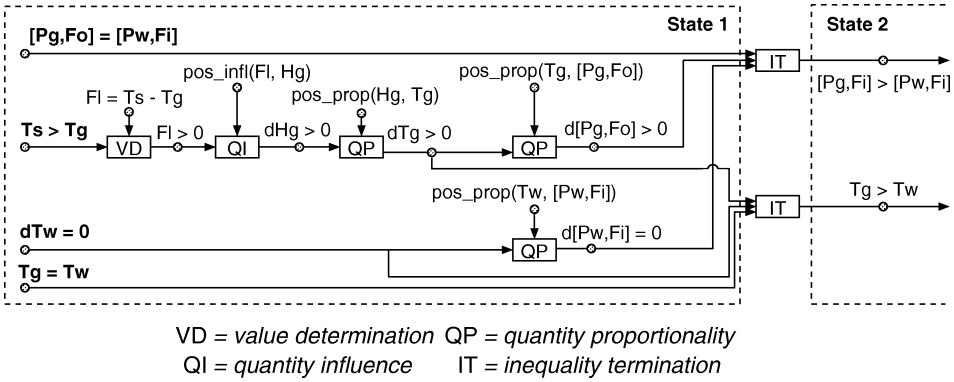

Fig. 17. Same model part after hiding of non-essential knowledge.

Between the heater and the gas in the container, a heat path exists. In the initial state of the system, the temperature of the gas and that of the outside world are given to be equal and the heater has been turned on, expressed by the fact that its temperature is higher than that of the gas. Furthermore, the temperature of the outside world is given not to change, and the piston is in its starting position. The model generation algorithm as described in Section 3.3 generates a base model for the behaviour of the piston system consisting of 824 components and 802 points. For the initial state and the transition to the second state, the most important part of the base model is shown in Fig. 16. At the left hand side, the bold face expressions like **Pos** $= s$ (the position of the piston is in the **s**tarting position) represent the inputs of the model, i.e., the information that is given to the learner, and hence can be assumed to be known. The movement of the piston is modelled as influenced by a 'move force' F_m, which is the difference between the outward force F_o and the inward force F_i. Because no friction is modelled, these two forces are equal to the pressure of the gas and the pressure of the surrounding world, respectively. What happens in the model part shown is that the temperature difference between the heat source and the gas ($T_s > T_g$), causes an increase in the pressure of the gas ($\delta P_g > 0$) and in the outward force ($\delta F_o > 0$). Because the temperature of the outside world is given to be steady ($\delta T = 0$), we can derive that the pressure of the gas becomes higher than the pressure in the outside world ($P_g > P_w$), and hence also the outward force becomes bigger than the inward force ($F_o > F_i$).

The first aggregation principle applied to the base model is that of hiding non-essential details (fully-corresponding quantities, submissive components, and inactive paths). The model that is generated on top of the base model is shown in Fig. 17. For hiding, the effect is particularly strong in the piston example because there are a number of quantities whose values never change.[9] For instance, both the temperature and the heat of the heat source (T_s and H_s), as well as those of the surrounding world (T_w and H_w), have the value plus in every behavioural state. Recall that in the base model, this kind of 'staying the same' from state to state is modelled by continuity components, which are abstracted from by the hiding algorithm. The hiding algorithm is also concerned with simplifying the model by merging

[9] In the base model, these values are indeed derived, but they are left out of Fig. 16 for clarity of presentation.

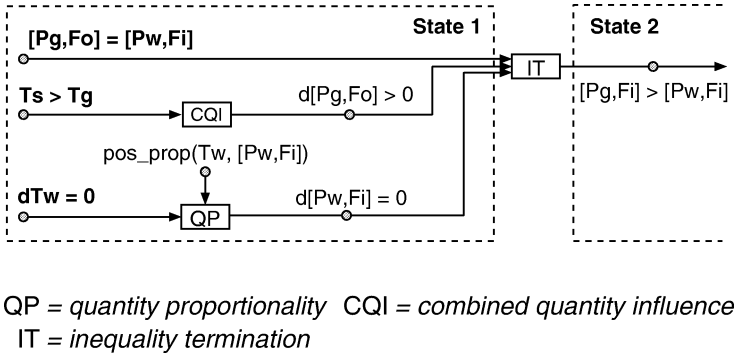

Fig. 18. Same model part after chunking.

quantities that are fully-corresponding. For example, the outward force and the pressure of the gas are modelled as fully corresponding, resulting in the combined quantity $[P_g, F_o]$, and also the volume of the gas and of the container ($[V_g, V_c]$). The latter combination is however removed because it does not matter for the reasoning in this state: the volume does not change, hence it has no effect on the pressure (cf. the submissive quantity proportionality component in Fig. 17). Component SIT is removed because it is a submissive termination.

Fig. 18 shows the next generated abstract model after chunking. Although in fact chunking is done in three phases, we only show the resulting model. The chain of reasoning steps from the model input $T_s > T_g$ to $\delta[P_g, F_o] > 0$ is chunked into one combined quantity influence component. As a result of this chunk, the inequality termination component deriving $T_g > T_w$ loses one input ($\delta T_g > 0$). This component is removed because $T_g > T_w$ is not used in any successor state; although it is a valid termination, it does not have any impact on the behaviour in the subsequent states, and hence is considered a "dead end".

Finally, grouping selects the different states and transitions, and creates one component for each (Fig. 19). As a result, the format of the data flow between the components changes from single expressions to sets of expressions representing state descriptions.

5. Diagnosing learner behaviour

Although the base models are defined in terms of components and connections for the sake of model-based diagnosis, some additional processing is needed to apply diagnostic

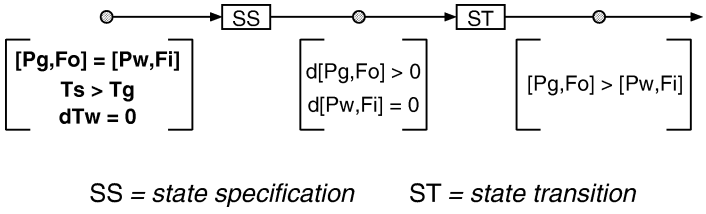

Fig. 19. Same model part after grouping.

techniques to these models. Firstly, some additional 'fine tuning' of the hierarchical models is required to befit the diagnostic task (Section 5.1). Secondly, we need a *prediction engine* that, given a set of observations, predicts the behaviour of the system to diagnose, in our case the learner's reasoning process (Section 5.2). Finally, the *diagnostic engine* of GDE needs a dedicated selection mechanism for obtaining additional observations (Section 5.3).

5.1. The role of inputs

In device diagnosis, 'inputs' are defined as the values that the diagnostician puts on a set of points in the device, in order to observe the values at another set of points, the 'outputs'. For models that we use, two different kinds of inputs exists. Firstly, *exercise inputs* are the expressions that are presented to the learner as part of the problem statement. For instance, when the learner is presented with the balance problem in Fig. 3, the exercise inputs are

$$W_l > W_r, \quad L_l < L_r, \quad V_l = V_r, \quad V_l > 0, \qquad V_r > 0.$$

Secondly, *support knowledge inputs* are all other inputs of the model, consisting of domain knowledge expressions that are used to perform inferences, such as causal dependencies and quantity spaces. For instance, the relation $dir\_corr(L, P)$ between the level and the pressure is a support knowledge input of the model.

The difference between the two types of inputs is problematic for diagnosis. The assumption made in (device) diagnosis that all model inputs are correct only holds for exercise inputs: the expressions explicitly presented to the learner can be assumed to be applied correctly. In contrast, support knowledge inputs *cannot* be assumed to be correct. For instance, learners may think that in the balance system the volume instead of the level determines the pressure. The problem is that a diagnosis that is defined in terms of defective components cannot discriminate between a support knowledge input that is not correctly known, and a reasoning step that cannot be executed: both will be traced to the component modelling the reasoning step. The solution to this problem is shown in Fig. 20. The correspondence between the level and the pressure in a contained liquid ($dir\_corr(L, P)$) is connected to the actual inference component by means of an intermediate *retrieval* component. We can now diagnose the fact that a learner does not know a support knowledge input as a *faulty retrieval component*. The retrieval inference can be interpreted as representing the action of retrieving a piece of knowledge from memory.

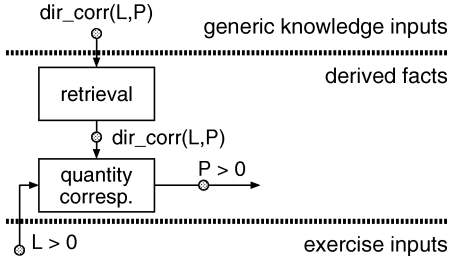

Fig. 20. Fallible domain knowledge inputs: retrieval components.

5.2. The prediction engine

The prediction engine used is similar to a constraint propagator, but instead of constraints it uses directed *rules* to propagate expressions through the model (cf. Section 3). For a component with three ports, three behaviour rules are needed to match the predictive power of a constraint. The complete set of component types as defined in Appendix A has 78 ports in total, and 40 behaviour rules are specified. Since the behaviour of the components is underspecified, the conflict set calculated by the prediction engine will not always be minimal. When the conflict set is non-minimal, this means that the set of resulting candidate diagnoses will also be non-minimal. However, the candidate set is still guaranteed to be complete: no valid candidates will ever be missed [30]. Because the hierarchical diagnostic models are small (typically less than 10 components), calculating larger candidate sets does not pose computational problems, although sometimes a larger number of probes will be needed to arrive at a satisfactory diagnosis.

5.3. The diagnostic engine

The GDE engine as introduced in Section 2.4.3 is used without significant modifications to calculate possible candidate diagnoses. The one important remaining bottleneck is which additional measurements should be taken to discriminate between candidate diagnoses. The specific nature of reasoning models requires a dedicated technique for selecting measure points. In this section, we discuss some properties of these models that can be exploited in developing criteria for measurement selection. Subsequently, we propose a method for determining measure points.

An important difference between models that we use and digital circuits lies in the nature of the components. In a digital circuit, two components of the same type may behave according to the same rules, but are still physically distinct instances. In reasoning models, this is not necessarily the case: one reasoning step applied correctly in one part of the model is very likely to behave correctly as well in another part: by their nature, different components of the same type are likely to fail collectively. A learner that does not know how to apply a quantity termination is likely to exhibit the same error (faulty quantity termination inference) at several places in the model. The only exception is formed by components of the retrieval type: here, different instantiations are indeed independent operations, because they refer to the retrieval of different knowledge facts. The error of not correctly 'retrieving' the relation between, for instance, level and pressure is usually not related to an incorrect retrieval of the negative influence of the, for instance, flow rate on the volume. This different nature of reasoning models is exploited by the diagnostic algorithm: the failure probability of a set of instances of the same component type is defined to be equal to that of a single component. Hence, a candidate consisting of four failing inequality correspondence components has the same probability as a single component candidate.

For retrieval components, it is possible to employ an additional heuristic in candidate discrimination: because most errors made in the experiment appeared to be caused by missing or confused domain knowledge [33], retrieval components can be assumed to have a higher *a priori* failure rate than inference components. In addition, *decomposable*

components have a higher *a priori* failure rate than individual base model components, because they incorporate a number of reasoning steps in the base model.

Important to note is that the use of *a priori* failure rate only depends on the type of component, and is independent of the specific contents of the simulation model. More in general, the diagnostic engine exploits only the structure of the model and the types of the components, and does not make any reference to the specific behaviour prediction that is modelled (see also Section 2.2).

The heuristic method used to find the split point in the set of candidate diagnoses is based on the same ideas as the *half split* criteria employed in for example SOPHIE-III [29] and FAULTY [5]. The half split approach aims at finding the point that optimally splits the set of components that contributes to a symptom: given the set of components *CpS* that contributes to a symptom, the *splitting factor* for a possible measure point p is defined as $|CpS_{bp} - CpS_{ap}|$, where CpS_{bp} is the subset of *CpS* contributing to the value of p ("before p") and CpS_{ap} the subset *CpS* not contributing to the value of p ("after p"). In our case, simply taking the difference in numbers of components does not work: because we do not make the single fault assumption, and also use different *a priori* fault probabilities, a candidate is no longer synonym with one component. Hence, we introduce the *weighted cardinality of a candidate*, facilitating the comparison of candidates. The weighted cardinality of a candidate expresses its probability in terms of the number and type of components it consists of.

The algorithm for candidate discrimination as shown below first checks whether the candidate set found so far is satisfactory *for the current hierarchical model*, according to the criterion of the educational system. If so, and further decomposition is possible, the decomposition algorithm proposes a new diagnostic model; if no further decomposition is possible, the diagnostic session ends (see below). Otherwise, the algorithm proceeds to determine the possible measure points. In step 2(b), the weighted cardinality of each candidate is calculated. The definition of a candidate's weighted cardinality embodies the different aspects discussed before: retrieval and decomposable components have a higher *a priori* failure rate and are counted individually. Components of the same type are counted only once. By its definition, the lower the weighted cardinality of a candidate, the higher its probability. Subsequently, we can map these weighted cardinalities on the individual components, yielding the *unnormalised probability of a component*. This probability expresses the different candidates that a component is part of: when a component belongs to more than one candidate, knowing its status will provide more information. Hence, the higher a component's unnormalised probability, the more important it is to focus the diagnostic process on this component (step 2(c)). The unnormalised probabilities of the components are used in calculating the splitting factor for each measure point (step 2(e)).

Algorithm: Candidate Discrimination

(1) If the candidate set *CaS* meets the criteria as set by the educational system, then go to the **decomposition** algorithm (Appendix B).
(2) Else, determine a measure point.
    (a) Collect the set of all possible measure points *MPS* by tracing backwards from the set of *symptoms* (i.e., observations that do not match the predicted value) through the model.

(b) For each candidate $Ca_i \in CaS$, calculate its *weighted cardinality* WC_{Ca_i}.
Let R be the number of retrieval components in Ca_i.
Let H be the number of decomposable components in Ca_i.
Let T be the number of other component *types* in Ca_i.
The weighted cardinality of a candidate is defined as $WC_{Ca_i} = 0.7 * R + 0.5 * H + T$.

(c) Let CpS be the set of components that contribute to the set of symptoms. Define the unnormalised probability of a component $Cp \in CpS$ to be $\sum 1/WC_{Ca_i}$ for all candidates Ca_i containing Cp.

(d) For each point p in MPS, let $CpS_{bp} \subset CpS$ be the set of components that contribute to the value of p and let $CpS_{ap} = CpS \backslash CpS_{bp}$.

(e) For each point p in MPS, calculate its splitting factor SF_p. Let UP_{bp} be the sum of the unnormalised probabilities of the components in CpS_{bp}, and UP_{ap} the sum of the unnormalised probabilities of the components in CpS_{ap}.
The splitting factor SF_p of measure point p is defined as $|UP_{bp} - UP_{ap}|$.

(f) Order the probe points in MPS according to their splitting factor SF_p: the best probe point is the one with the smallest value for SF.

(3) Execute a probe by outputting the ordered list of probe points, and wait for the set OBS of returned expressions.

(4) Call the GDE **conflict recognition** algorithm (Appendix B) with the set OBS.

The definition of the weighted cardinality of a candidate includes some numerical interpretations of qualitative observations that may prove to be non-optimal. For example, the fact that a retrieval component is counted as 0.7 is a somewhat arbitrary quantification of the observation that more errors are made in the domain knowledge than in the reasoning knowledge. Because of the small number of components that is diagnosed at once in the hierarchical model, the impact of such choices will be relatively small.

A major advantage of the discrimination algorithm is that the computational costs are low because no behaviour propagation is involved. However, the final diagnosis may not always be found in the minimum number of probes possible: the algorithm does not use estimations about the total number of probes needed to pin down the final diagnosis, as is for instance done in [28]. Apart from the fact that in general estimations of the total number of probes needed are not necessarily cost-effective [29], a more specific reason exists for not using such estimations: because the hierarchical models ensure that the diagnostic models are small, the number of possible probe points is generally low. Hence, the impact of a suboptimal probe point selection is limited.

As defined in step (3), the candidate discrimination algorithm does not deliver one probe point, but a list of possible probe points ordered to their discriminating power (i.e., their splitting factor). Although the diagnostic machinery can reason about the expected *results* of a certain probe point, it cannot determine the *costs* of a specific probe within the current educational context. As a result, the most effective probe suggested may be very expensive, in the sense that it does not fit in with the current dialogue. In this case, the educational system may select another probe point from the list.

The diagnostic method presented above works within a model at one level of abstraction. Through *decomposition*, a diagnosis found on one level of abstraction can be refined

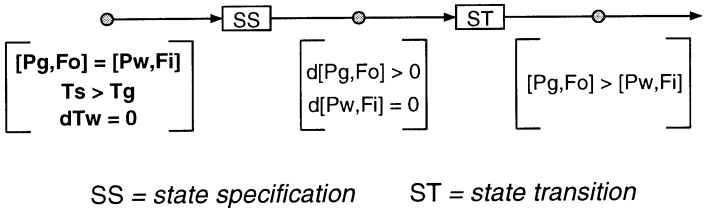

Fig. 21. Top-level diagnostic model.

by considering the related components at a lower level. Especially if multiple faults are possible, it is difficult to ensure soundness and completeness of the diagnostic process when hierarchical decomposition is applied [57]. In our models we cannot expect soundness or completeness: apart from the possibility of multiple faults, there are also different kinds of aggregations with different properties. We take a practical stance: within a specific model, we diagnose until the external criteria for what constitutes a satisfying diagnosis are met. If not already at the lowest hierarchical level, the component with the maximum unnormalised probability is decomposed, and diagnosis is resumed in the resulting lower-level model. If the diagnostic algorithm returns an empty set of candidates (i.e., the decomposed model does not reveal a conflict), the decomposition is undone and the next most probable component is selected for decomposition. This last step is only needed to account for the (undesirable) situation in which the external criterion for a satisfactory diagnosis is too weak. Hence, the candidate discrimination algorithm may call for decomposition with a too large candidate set: the component first selected may not belong to the correct minimal diagnosis.

5.4. An example diagnosis

This section exemplifies the use of the diagnostic algorithms in an educational setting. We continue the piston example as used in Section 4.4. The top-level model from Fig. 19 for the first state and transition is repeated in Fig. 21.[10] The learner is presented with a description and a picture of the initial behavioural state of the piston system. The questions asked are multiple choice. Suppose the strategy for subject matter sequencing that is used by the educational system is to follow the top-level model and ask for each change (i.e., each output of a transition component).[11] As a result of this strategy, the first question asked is about the output of the transition component ST. The only expression available is the inequality between the combined pressure/force quantities of the world and the gas. For convenience, in the remainder of this section we will only refer to pressure, both in the questions and in the figures. However, the educational system has the possibility to

[10] Recall from Section 4.3 that the input of a state transition also contains the input expressions of the preceding state specification. These are left out for reasons of presentation.

[11] Additional control is needed when alternative transitions and/or cycles exist, but this is not essential to the diagnostic approach.

talk about force instead of pressure if that would fit in better with the current educational context.

Probe 1.
Teacher: The pressure of the gas is initially equal to the pressure of the outside world ($P_g = P_w$).
What do you think about this pressure ratio in the *next* behavioural state?
a. $P_g < P_w$;
b. $P_g = P_w$;
c. $P_g > P_w$.
Learner: b. $P_g = P_w$.

The answer is wrong, so one or more components in Fig. 21 must be erroneous. Because no observations are known about the point between SS and ST, the top-level focusing algorithm asks for additional observations. At this point, two expressions can be measured: $\delta P_g > 0$ or $\delta P_w = 0$. Suppose the educational system chooses to ask only the first one.

Probe 2.
Teacher: Is the pressure of the gas initially:
a. increasing;
b. steady;
c. decreasing.
Learner: a. increasing.

According to the top-level focussing heuristic, the diagnostic system assumes that an error has been made in the transition component: an expression in the input is measured to be correct, and one in the output is measured to be incorrect. Therefore, the transition component is decomposed, resulting in the situation in Fig. 22. This decomposition does not yield a larger number of components, but only splits the compound expression into three new ones. Because we do not have an observation for the lowest input port of IT-2 (δP_w), no conflict is found: the prediction engine will derive the incorrect expression $\delta P_w > 0$ for this port, but there is no observation to conflict with this expression. As a

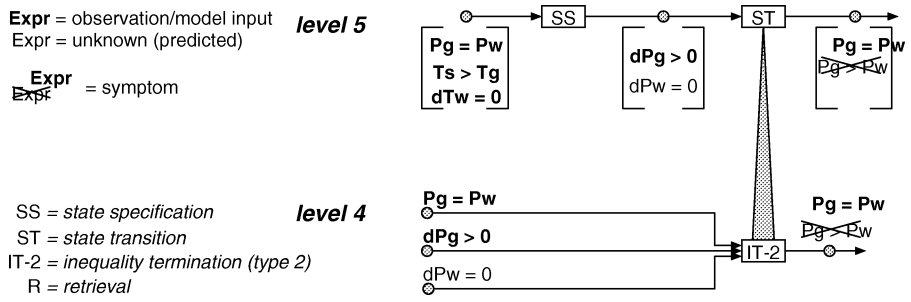

Fig. 22. Decomposition of the state transition component.

result, the diagnoser returns with an empty conflict set, and the educational system is asked for new observations at the top level. The only resulting expression to ask for is $\delta P_w = 0$.

Probe 3.
Teacher: Is the pressure of the world initially:
a. increasing;
b. steady;
c. decreasing.
Learner: b. steady.

Because the answer is correct, the same decomposition of ST is executed. This time, the new observation results in the single-fault diagnosis [IT-2]: all inputs are either given ($P_g = P_w$) or observed to be correct ($\delta P_g > 0,\ \delta P_w = 0$), and the output is observed to be faulty ($P_g = P_w$).

As a second example, consider the case in which the learner gives another answer in Probe 2.

Probe 2[b].
Teacher: Is the pressure of the gas initially:
a. increasing;
b. steady;
c. decreasing.
Learner: b. steady.

The top-level focussing algorithm now restarts the search for the initial focus from the point between SS and ST, and finds that the error must be in SS: all inputs are known to be correct, because they are given, and the output contains an incorrect observation. Hence, the state specification component is decomposed as shown in Fig. 23.

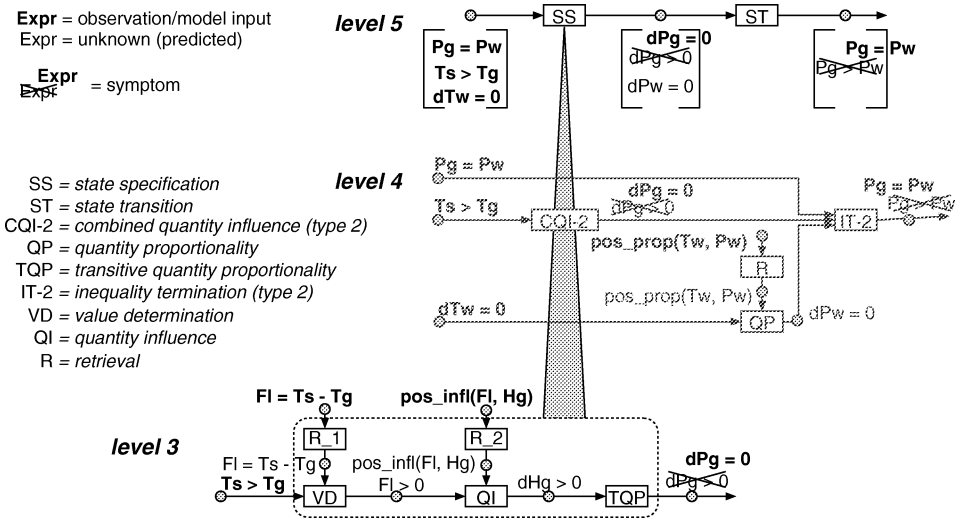

Fig. 23. Decomposition of the state specification component.

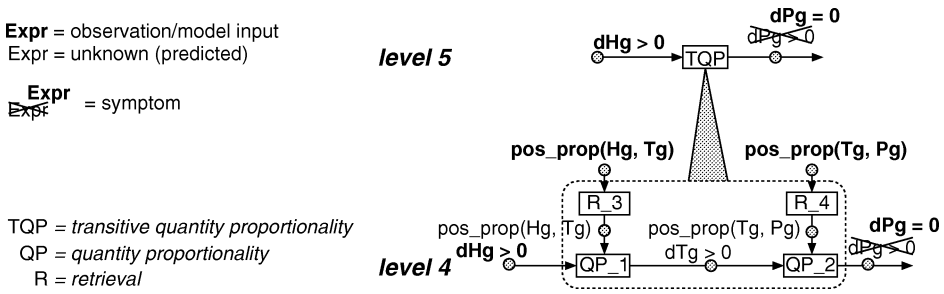

Fig. 24. Decomposition of the transitive quantity proportionality component.

In this case, the decomposition yields an immediate single-fault diagnosis [CQI-2] at level 4. Because CQI-2 is an aggregated component, the decomposition algorithm (see Appendix B) is called. This results in the level 3 model shown in the lower part of Fig. 23. The diagnostic algorithm is now called on this five component model. One conflict is found, namely $\langle \mathsf{R}_1,\ \mathsf{R}_2,\ \mathsf{VD},\ \mathsf{QI},\ \mathsf{TQP}\rangle$, resulting in the single-fault candidates $[\mathsf{R}_1]$, $[\mathsf{R}_2]$, [VD], [QI], and [TQP]. There are four possible measure points that provide information for discriminating between these components, labeled $Fl = T_s - T_g$, $Fl > 0$, *pos_infl*$(Fl,\ H_g)$, and $\delta H_g > 0$. Because each of the five components in the submodel is a single-fault candidate, the unnormalised probability is $1/0.7 = 1.43$ for the two retrieval components R_1 and R_2, $1/0.5 = 2$ for the decomposable component TQP, and 1 for the base model components VD and QI. Calculation of the splitting factor of these measure points yields the smallest value for the point $Fl > 0$: $|(1.43 + 1) - (1.43 + 1 + 2)| = 2$. Hence, the best point to probe is to ask for the value of the flow rate *Fl*.

Probe 3[b].
Teacher: What is the direction of the heat flow between source and gas?
a. from source to gas;
b. from gas to source;
c. there is no heat flow between source and gas.
Learner: a. from source to gas.

The new candidate set consists of $[\mathsf{R}_2]$, [QI], and [TQP]. Calculating the splitting factor for the two remaining measure points *pos_infl*$(Fl,\ H_g)$ and $\delta H_g > 0$ now yields $|1.43 - (1 + 2)| = 1.57$ and $|(1.43 + 1) - 2| = 0.43$, respectively. Now $\delta H_g > 0$ is measured.

Probe 4[b].
Teacher: Is the heat of the gas initially:
a. increasing;
b. steady;
c. decreasing.
Learner: a. increasing.

The single-fault diagnosis found now, [TQP], results in a new decomposition, as depicted in Fig. 24. The diagnoser returns a candidate set consisting of $[\mathsf{R}_3]$, $[\mathsf{QP}_1]$, $[\mathsf{R}_4]$, and $[\mathsf{QP}_2]$.

The measure point $\delta T_g > 0$ exactly splits the set of candidates in two, as is reflected by its splitting factor $|(1.43+1)-(1.43+1)|=0$. Hence, the next probe concerns δT_g.

Probe 5[b].
Teacher: Is the temperature of the gas initially:
a. increasing;
b. steady;
c. decreasing.
Learner: a. increasing.

Because the answer is correct, two candidates [R_2] and [QP_2] remain, and one possible probe point *pos_prop*(T_g, H_g).

Probe 6[b].
Teacher: What is the relation between the temperature and the pressure of the gas in the current state?
a. if the temperature increases, then the pressure increases;
b. if the temperature increases, then the pressure decreases;
c. if the temperature increases, then this does not affect the pressure.
Learner: a. if the temperature increases, then the pressure increases.

The final diagnosis found is [QP_2]: the learner does know the relation between temperature and pressure, but did not apply it here.

The interaction traces reflect diagnostic sessions in which a maximally detailed diagnosis is determined using a minimal amount of information. In a full-fledged educational system, other functional components are available to provide a richer interaction, such as a discourse planner [80]. In addition, a number of observations may be already available from the learner model, which can be used both to improve the dialogue and to reduce the number of probes. The educational system does not need to proceed until one single fault is located in the base model, but may decide to start explaining a larger part of the process after some probes.

6. The STAR^{light} prototype system

To assess the feasibility of the diagnostic process and the usefulness of the aggregated subject matter models, the STAR^{light} prototype system[12] was developed and try-out sessions with learners were conducted. The design of the STAR^{light} system concentrated on the main research topics, namely subject matter model generation and diagnosis. Both components were fully implemented according to the techniques and algorithms described in this article. For the other components of the architecture, which was shown in Fig. 2, we implemented 'light' versions.

[12] The acronym STAR stands for System for Teaching About Reasoning. Because the prototype only implements the subject matter model and the diagnostic function of the STAR system in detail, it is christened STAR^{light}.

The explanation generator in the prototype is based on explaining individual components. For each component type, a generic explanation frame can be constructed that complies with the current situation, and this frame is filled in with the terms in the model using a set of domain-specific verbalisations. The question generator works similarly, and generates multiple choice questions about specific points in the model.

Although in the implementation hardly any effort was made to optimise the performance of the system, it is nevertheless adequate for real-life applications. On a 200 MHz Pentium-Pro platform under Linux, the diagnostic component returns a candidate set mostly within one second. The longest calculation time needed in the try-out sessions was four seconds. The domain-specific verbalisations and graphics are as yet only implemented for the balance domain. Hence, the try-out sessions of the system were conducted using this system. For larger predictions, the performance is expected to be similar: as a result of the aggregations used, the size and complexity of the diagnostic models does not increase with larger simulations. The only increase is in the simulation and model generation process, but these can be processed in the background.

An example of the interface screen is shown in Fig. 25. In the upper half of the screen, the behavioural states of the balance system are shown. The pictures are generated from the subject matter model; a set of quantitative measures is added to define the actual size of the levels and the widths of the water columns on the screen. The lower half of the screen displays the questions and explanations provided by the system. At the bottom, buttons are provided for selection of an answer, as well as a 'give me a hint' button that provides a help text in a separate window. This help text is also constructed by the explanation generator.

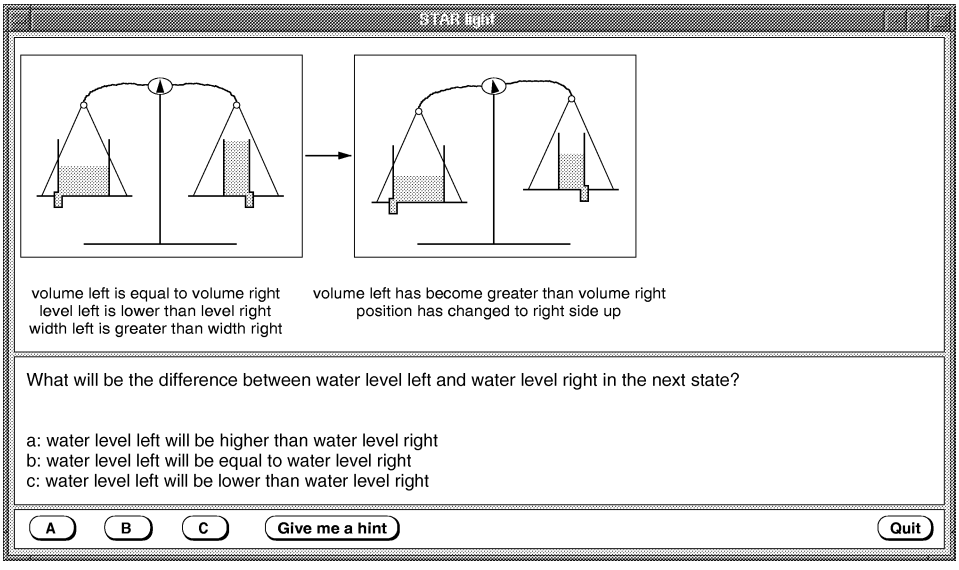

Fig. 25. A screen dump of the interface.

6.1. Try-out sessions

We conducted a small number of try-out sessions to determine whether the STAR^{light} system is useable in an educational setting. Because the STAR^{light} system has been developed to demonstrate the diagnostic framework developed, we did not try to evaluate it as a full-fledged educational system.

In the try-out sessions, the subjects were asked to do a series of four balance prediction exercises. The think-aloud method [71] was employed to provide information about the subject's reasoning behaviour. There were two separate sessions. In the first session, four subjects participated, all being third or fourth year university students who had some background in knowledge-based educational systems and the subject matter (i.e., qualitative behaviour analysis). These subjects will be referred to as 'advanced subjects'. In the second session, there were five subjects from different backgrounds, who had no experience with either educational systems or the subject matter. We refer to these subjects as 'novice subjects'. The experiment took about 30 minutes for each subject. Afterwards, the subjects were asked a number of questions about their experiences. These questions were aimed at the explanations of the final diagnoses and at the probe questions. The subjects were asked whether they thought that the system correctly determined the mistake they made, and whether the questions asked by the diagnoser were appropriate, in a logical order, and not too large in number.

Each of the nine subjects performed the same four prediction exercises about the balance system. In total, 31 diagnostic sessions were run and 707 questions were answered. Novices appeared to make about twice as many errors as advanced subjects. For most cases the number of probes that was needed, to arrive at a satisfactory diagnosis, varied from one to three. In eight diagnostic sessions, more probes were needed. The longest sequence of probes was eight, which occurred twice. The number of probes per diagnoses was on average the same for novices and advanced subjects. Also, the variation in the types of the components found as the final diagnoses (e.g., retrieval components versus other component types) was not significantly different in both groups.

6.2. An example diagnosis

A worked-out example is provided of a diagnosis session that occurred in the try-out sessions. The example is taken from the first session with advanced subjects. A part of the interaction, combined with the think-aloud remarks made by the subject, is shown below.[13] The dialogue starts when the subject is asked to predict the next behaviour state, starting from the balance situation shown in Fig. 3. This situation is equal to the leftmost state shown in Fig. 25.

1. What will be the difference between volume left and volume right in the next state?
2. *Hey, look, the volumes are equal. I didn't take notice of that.*
3. *Yes, I think they stay equal?*

[13] Lines with plain text are generated by the STAR^{light} system. Lines in italic starting with *a*:, *b*: or *c*: are answers given by the learner. Think-aloud remarks are in italic with indent.

4. *b: volume left will be equal to volume right*
5. That's not correct.
6. Before we go on with the prediction, I would like to know what you think about the present situation.
7. Please answer the following question:
8. What do you know about the difference between the decrease in volume left and the decrease in volume right?
9. *Volume left decreasing slower than volume right?*
10. *No, at the same speed I would say.*
11. *b: volume left is decreasing at the same speed as volume right*
12. That's not true.
13. Please answer the following question:
14. What do you know about the difference between outflow left and outflow right?
15. *Well I would say that the outflow is equal, but that is now when the pressure [incomprehensible]*
16. *Ah! Is that it? The outflow left is smaller*
17. *c: outflow left is smaller than outflow right*
18. That's right.
19. Please answer the following question:
20. which quantity causes a change in volume?
21. *The outflow, or the pressure, yes the outflow*
22. *c: outflow*
23. That's right.
24. I think you don't fully understand the following:
25. the difference between outflow left and outflow right has a direct influence on the difference between the decrease in volume left and the decrease in volume right.
26. This means that because outflow left is smaller than outflow right, volume left is decreasing slower than volume right.
27. Let's continue with the exercise where we left off.
28. *All right, interesting...*

To explain the diagnostic process underlying this interaction, consider Fig. 26. The answer given in line 4 supplies the observation $V_l = V_r$ in the output of the state transition component ST. ST models the transition from the first to the second state at the highest hierarchical level. In the prototype system, the subject matter sequencing is simplified to only asking the output of each subsequent transition component. Hence, a conflict at the highest level results in a decomposition of *two* components: the preceding transition component plus the previous specification component (for more details, see [31]). Hence, SS and ST are decomposed into the seven-component model depicted. The first call to the diagnoser delivers one conflict: ⟨CII, IT⟩, and hence two diagnoses [CII] and [IT]. The combined inequality influence CII is a decomposable component that summarises the calculation of a derivative inequality (in this case, the ratio of the decreases in volume $\delta V_l > \delta V_r$) from an (in)equality (the ratio of the water levels $L_l < L_r$). The inequality termination IT determines the new inequality between the values in the next state. The only probe point that yields information about the candidates [CII] and [IT] is in between

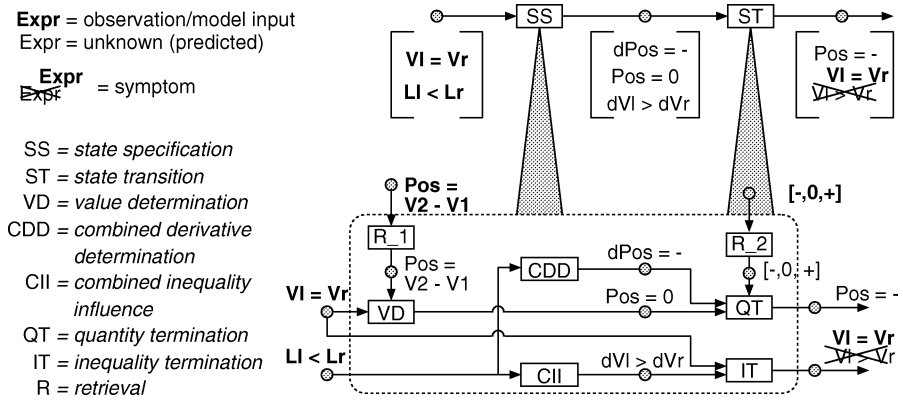

Fig. 26. First diagnostic cycle.

these components. Hence, a question is asked about the inequality between the derivatives of the volumes ($\delta V_l > \delta V_r$, line 8). The answer given in line 11 is incorrect, yielding a single-fault diagnosis [CII]. Because this is a higher-level component, it is decomposed into a lower-level model as shown in Fig. 27. The next diagnostic cycle yields one conflict $\langle$TIC, R_3, II$\rangle$ and three candidates [TIC], [R_3], and [II]. For the two existing measure points, the splitting factors are determined by the discrimination algorithm. For the point *neg_infl*(Fl, V), the splitting factor is calculated on the basis of the *a priori* failure rates of retrieval component R_3 (0.7), decomposable component TIC (0.5), and 'normal' base model component II (1): $|1/0.7 - (1/0.5 + 1/1)| = 1.57$. For $F_l < F_r$, the splitting factor is $|1/0.5 - (1/0.7 + 1/1)| = 0.43$. $F_l < F_r$ has the lowest value, and thus the highest discriminating power. Hence, this one is questioned (line 14) and answered correctly (line 17). This results in the new conflict $\langle R_3,$ II$\rangle$ and two candidates [R_3] and [II]. The last probe on *neg_infl*(Fl, V) in line 20 delivers the inequality influence component II as a final single-fault diagnosis. In lines 24–26, an explanation is generated for this component.

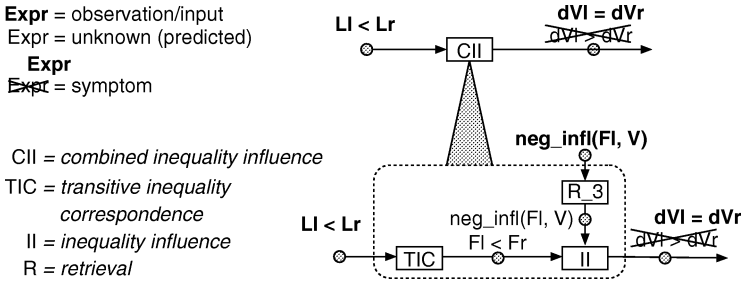

Fig. 27. Second diagnostic cycle.

6.3. Evaluation

According to the interviews held with the subjects after the experiment, especially advanced subjects were satisfied about the prototype's interaction. The amount of questions was considered reasonable, the subject of the questions to the point, and the order in which the questions were asked occurred natural to them. In general, they had the idea that the diagnoses were correct and the explanations useful, although sometimes "outdated": the probe questions sometimes triggered self-repair by the subjects, and hence the ultimate diagnosis pointed out an already fixed problem (see also Section 8).

Novices reported problems with the sometimes rather cryptic questions and the specific, one-component based explanations. Clearly, a more advanced question and explanation generator is required to communicate with novices. Furthermore, the balance system may not be very suitable for novices, because it involves rather complicated inequality reasoning. As will be discussed in the next section, starting with simpler systems (e.g., one emptying container) may yield better results for novices.

7. Related work

As indicated in Section 2.4, we propose a novel, generic approach to diagnosis of learner behaviour based on existing techniques from the field of model-based reasoning. Whereas we discussed qualitative reasoning and model-based diagnosis, as well as earlier approaches to educational diagnosis, we did not yet discuss how the aggregation techniques relate to existing work on abstraction in both model-based reasoning and theories on education

An alternative for the aggregation procedures presented in this article would be to derive the abstract models by generating more abstract simulations of the intended behaviour in the first place [37]. The base model generation algorithm can then be applied to each of these models to generate the related hierarchical layer in the subject matter model. In this alternative approach the actual abstraction process has moved to the scenarios and the library of model fragments. In fact, within the field of qualitative reasoning, substantial research has been done on how the *generation* of qualitative simulation models can be adapted, e.g. [54,61,66,73,74]. The motivation for these adaptations varies from removing irrelevant details, or spurious behaviours, to tailoring for a specific task. An important advantage of abstraction before simulation is that often the number of states that is generated by the simulator can be decreased; this is precisely the aim when trying to avoid spurious behaviours. However, in our research we have the additional goal of exploiting the abstractions in an educational context. We therefore need to have the relations between the different abstraction models explicitly available. If we generate the models separately, from different scenarios, we do not have explicit knowledge about the connections between the components in the different models.

Some research has been done on summarising the results of qualitative simulations, i.e. reformulating the output after it has been generated. In [42], an approach is described that uses a set of heuristics to collapse chains in a causal order graph of quantity relations for the purpose of simplifying explanations. The major difference with our approach lies in

the application of the technique: while they focus on explanations, we focus on teaching a *reasoning process*. As such, we do not aggregate the relations in the causal order graph itself, but a model that represents the reasoning steps that can be made with these relations. This way, we can locate those reasoning steps that a student did not yet master, and not only the causal relations they do not know. Most other approaches that are aimed at simplifying simulation outputs operate again at the level of states. An example is [55], where complex behaviour graphs are abstracted by reducing the number of states to be considered. This approach is complementary to ours. We take the number of states and state transitions as a given, and try to abstract from less relevant details in the reasoning that is done 'within' these states and transitions.

The aggregation procedures presented in this article hide details when mapping the current level to the next higher level. The idea of exploiting models of differing complexity in teaching has been investigated in [76]. [14] *Model Progression Theory* proposes a set of models to be used in educational systems. They define three dimensions on which these models may vary: *perspective*, *order*, and *degree of elaboration*. The key idea is that learning can be optimally supported when different models are presented in succession, starting from the simplest one. The aims of model progression are clearly different from ours, and therefore the techniques are complimentary. Model progression is primarily concerned with subject matter sequencing and supporting the learning process at a more global level. Hence, where model progression is about selecting the right model in a progressive sequence, we take a specific simulation model to start with as a given. In fact, the evaluative remarks in Section 6.3 already indicated that for novice learners a 'model progression-like' approach can be very beneficial. Note however that the construction of a set of models according to the dimensions specified by causal model progression is as yet not an automated process (see [34] for ideas on how to automate model progression).

Whereas model progression concentrates mainly on acquiring new knowledge, there are also a number of theories that model learning in terms of combining and improving existing knowledge. In SOAR [52], a general architecture for modelling intelligent behaviour, an important learning operator is *chunking*: if a procedure (consisting of a set of inferences) is processed once, then often one abstract inference can be compiled to replace future processing of the procedure [53]. SOAR's learning-by-chunking mainly increases efficiency. *Knowledge Compilation* [2] is similar to chunking, in that it models the process of constructing domain-specific shortcuts for general (inefficient) reasoning paths during learning. In the ACT* and ACT-R theory, knowledge compilation is viewed as an important learning mechanism in skill acquisition [1]. An important difference between these techniques and the aggregation principles described in this article is that we do not aim at modelling efficient learning over cases, but at supporting diagnosis in one specific case. The goal of both chunking and knowledge compilation is to facilitate easier and more efficient reasoning in future occurrences of the same problem. We do not generalise over cases (i.e., predictions), but only compact inference sequences within one case. A second difference is the fact that our aggregation approach covers *all possible* correct reasoning paths of a specific case. In compilation and also explanation based generalisation [56] chunking is only over *one* specific reasoning trace per case.

[14] In [64], multiple models are employed in a related way, although with a different theoretical footing.

8. Conclusions

In this article we showed that model-based techniques can be applied successfully to solve outstanding problems in research on the use of AI technology for educational purposes. In particular, we employed qualitative simulation and model-based diagnosis to represent and analyse the reasoning behaviour of a learner when predicting the behaviour of a device. The main advantage of this approach is that it provides a *generic method* for the task of cognitive diagnosis, which as of yet has mostly been based on predefined, domain-specific bug catalogues. Qualitative simulators provide a means for automatically generating the behaviour of a device, and this output was shown to be convertible into a *base model* that can be used as the backbone of an educational system: the base model represents all the correct facts and reasoning steps that are required for a correct prediction of behaviour. Being based on earlier experimental research about human problem solving behaviour, the knowledge represented in the base model can be considered didactically relevant: both the type and the grain size of the reasoning steps in the model correspond to how teachers and learners communicate about device behaviour.

The base model adheres to the representational constraints of model-based reasoning: each reasoning step is defined as an inference component in the model. Hence, diagnostic techniques such as GDE can be applied. Accordingly, the diagnostic task is defined as determining those reasoning steps that the learner cannot have applied correctly given the observations. The important consequence of this definition is that we do not try to grasp the learner's mental model, nor that we force the learner into a predefined reasoning trajectory: we do not assume that the reasoning model is a cognitive model of what goes on in the mind of the student. We do assume however that the model is didactically plausible: it represents knowledge that the student should acquire in the learning process. The role of diagnosis is not to detect "mind bugs", but to detect and locate *behavioural* deviations. The results of the diagnosis are used to focus the dialogue with the student on those topics that correspond to erroneous behaviour.

The application of GDE to the base models required two issues to be addressed. Firstly, a new technique for measurement selection had to be developed that accounts for the selection of educationally relevant probing points (i.e., subjects for questions). Secondly, the base models were too large and too loosely structured to apply model-based diagnostic techniques directly in an efficient and effective manner. We therefore developed a structuring mechanism that automatically transforms the base model into a hierarchy of aggregated models by removing irrelevant detail and by chunking chains of causal inference steps. The resulting structured subject matter model does sufficiently enable the diagnostic process to produce useful diagnostic results in a real-time environment.

An experiment was conducted with learners that showed the utility of the approach presented in this article. We believe the STAR framework to be particularly useful for the development of educational environments that stimulate the learner's *self-repair* capabilities. The experiment showed that by focusing on behaviour errors in a systematic way, the learner's ability to self-repair is stimulated. Such a focus on learning from errors requires a view on education that is not commonly practiced by human teachers: they appear to rely mainly on pattern recognition on the basis of known misconceptions, rather than on detailed diagnosis [13,51]. On the one hand, this is influenced by traditional

educational views on errors: "School teaches that errors are bad; the last thing one wants to do is to pore over them, dwell on them, or think about them" [59]. But more importantly, detailed structured diagnosis is often computationally infeasible for human teachers.

The prototype system presented is only a partial implementation of the complete framework we envision. In the near future, other components of the architecture will be researched in more detail, hopefully resulting in a complete set of truly model-based techniques that can support education in an articulate and yet generic way.

Acknowledgement

The authors would like to thank David Rühl for his contributions to the implementation of the STAR^{light} prototype system. Richard Benjamins, Frank van Harmelen, Remco Straatman and Radboud Winkels provided valuable comments on earlier drafts of this article.

Appendix A. Model component definitions

For the 10 base model component types, the definitions are provided below. Note that two versions exist for inequality terminations, value determinations and derivative determinations, depending on the number and type of inputs used for the inference (see also [31]). When different possibilities for one expression are given, such as in $[\delta A = -/0/+]$, the positional equivalent expression should be chosen in other parts of the rule. The notations *succ*(*Val*) and *pred*(*Val*) stand for the successor and predecessor of *Val* in its quantity space, respectively.

Quantity Correspondence

ports: In = quantity value, Sup = value correspondence, Out = quantity value
example: "The level is positive, therefore the pressure is positive as well"
$(L > 0 \,\&\, \text{dir\_corr}(L, P) \rightarrow P > 0)$
forward behaviour rules: In & Sup → Out
IF In = $[A = \text{Val}]$ & Sup = $[\text{corr}(A, B)]$ THEN Out = $[B = \text{Val}]$
IF In = $[A = \text{Val}]$ & Sup = $[\text{corr}(B, A)]$ THEN Out = $[B = \text{Val}]$
IF In = $[A = \text{Val}]$ & Sup = $[\text{dir\_corr}(A, B)]$ THEN Out = $[B = \text{Val}]$
IF In = $[A = \text{Val}_A]$ & Sup = $[\text{v\_corr}(A, \text{Val}_A, B, \text{Val}_B)]$ THEN Out = $[B = \text{Val}_B]$
IF In = $[A = \text{Val}_A]$ & Sup = $[\text{v\_corr}(B, \text{Val}_B, A, \text{Val}_A)]$ THEN Out = $[B = \text{Val}_B]$
IF In = $[A = \text{Val}_A]$ & Sup = $[\text{dir\_v\_corr}(A, \text{Val}_A, B, \text{Val}_B)]$
THEN Out = $[B = \text{Val}_B]$
backward behaviour rules: Out & Sup → In

Quantity Proportionality

ports: In =quantity derivative, Sup = proportionality, Out = quantity derivative
example: "The level decreases, therefore the pressure is decreasing as well"
$(\delta L < 0 \,\&\, \text{pos\_prop}(L_l, P_l) \rightarrow \delta P_l < 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [\delta A = -/0/+] \,\&\, \mathsf{Sup} = [\text{pos\_prop}(A, B)]$ THEN $\mathsf{Out} = [\delta B = -/0/+]$
IF $\mathsf{In} = [\delta A = -/0/+] \,\&\, \mathsf{Sup} = [\text{neg\_prop}(A, B)]$ THEN $\mathsf{Out} = [\delta B = +/0/-]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$
$\mathsf{In} \neq 0 \,\&\, \mathsf{Out} \neq 0 \rightarrow \mathsf{Sup}$

Quantity Influence

ports: In = quantity value, Sup = influence, Out = quantity derivative
example: "There is a [positive] flow, so the volume decreases"
$(Fl > 0 \,\&\, \text{neg\_infl}(Fl, V) \rightarrow \delta V < 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [A >/=/< 0] \,\&\, \mathsf{Sup} = [\text{pos\_infl}(A, B)]$ THEN $\mathsf{Out} = [\delta B = +/0/-]$
IF $\mathsf{In} = [A >/=/< 0] \,\&\, \mathsf{Sup} = [\text{neg\_infl}(A, B)]$ THEN $\mathsf{Out} = [\delta B = -/0/+]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$
$\mathsf{In} \neq 0 \,\&\, \mathsf{Out} \neq 0 \rightarrow \mathsf{Sup}$

Quantity Termination

ports: In = quantity value, derivative, Sup = quantity space, Out = quantity value

example: "The volume goes down to zero, so it will become empty"
$(V > 0, \delta V < 0 \,\&\, [0, +] \rightarrow V = 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In1} = [A = \text{Val}] \,\&\, \mathsf{In2} = [\delta A = +/-] \,\&\, \mathsf{Sup} = \mathsf{QS(A)}$
THEN $\mathsf{Out} = [A = \text{succ(Val)/pred(Val)}]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \,\&\, \mathsf{In1} \rightarrow \mathsf{In2}$
$\mathsf{Out} \,\&\, \mathsf{Sup} \,\&\, \mathsf{In2} \rightarrow \mathsf{In1}$

Inequality Correspondence

ports: In = value inequality, Sup = value correspondence, Out = value inequality
example: "The level is higher [at the right], [therefore] the pressure as well"
$(L_l < L_r \,\&\, \text{dir\_corr}(L, P) \rightarrow P_l < P_r)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [A_1 </=/> A_2] \,\&\, \mathsf{Sup} = [\text{corr}(A, B)]$ THEN $\mathsf{Out} = [B_1 </=/> B_2]$
IF $\mathsf{In} = [A_1 </=/> A_2] \,\&\, \mathsf{Sup} = [\text{corr}(B, A)]$ THEN $\mathsf{Out} = [B_1 </=/> B_2]$
IF $\mathsf{In} = [A_1 </=/> A_2] \,\&\, \mathsf{Sup} = [\text{dir\_corr}(A, B)]$ THEN $\mathsf{Out} = [B_1 </=/> B_2]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$

Inequality Proportionality

parts: In = derivative inequality, Sup = proportionality, Out = derivative inequality

example: "The level decreases faster at the left, so does the pressure"
$(\delta L_l < \delta L_r \,\&\, \text{pos\_prop}(L, P) \rightarrow \delta P_l < \delta P_r)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [\delta A_1 </=/> \delta A_2] \,\&\, \mathsf{Sup} = [\text{pos\_prop}(A, B)]$
THEN $\mathsf{Out} = [\delta B_1 </=/> \delta B_2]$
IF $\mathsf{In} = [\delta A_1 </=/> \delta A_2] \,\&\, \mathsf{Sup} = [\text{neg\_prop}(A, B)]$
THEN $\mathsf{Out} = [\delta B_1 >/=/< \delta B_2]$
backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$

Inequality Influence

parts: In = value inequality, Sup = influence, Out = derivative
example: "The right flows faster, so [the volume] decreases faster"
$(Fl_l < Fl_r \,\&\, \text{neg\_infl}(F, V) \rightarrow \delta V_l > \delta V_r)$
forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [A_1 </=/> A_2] \,\&\, \mathsf{Sup} = [\text{pos\_infl}(A, B)]$
THEN $\mathsf{Out} = [\delta B_1 </=/> \delta B_2]$
IF $\mathsf{In} = [A_1 </=/> A_2] \,\&\, \mathsf{Sup} = [\text{neg\_infl}(A, B)]$
THEN $\mathsf{Out} = [\delta B_1 >/=/< \delta B_2]$
backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$

Inequality Termination (type 1)

ports: In = value inequality, derivative inequality $\mathsf{Sup} = \emptyset$, Out = value inequality
example: "The level is higher [at the right], but also goes down faster, that means they become equal"
$(L_l < L_r, \delta L_l > \delta L_r \,\&\, \text{—} \rightarrow L_l = L_r)$
forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In1} = [A_1 =/> A_2] \,\&\, \mathsf{In2} = [\delta A_1 < \delta A_2]$ THEN $\mathsf{Out} = [A_1 </= A_2]$
IF $\mathsf{In1} = [A_1 </= A_2] \,\&\, \mathsf{In2} = [\delta A_1 > \delta A_2]$ THEN $\mathsf{Out} = [A_1 =/> A_2]$
backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{In1} \rightarrow \mathsf{In2}$
$\mathsf{Out} \,\&\, \mathsf{In2} \rightarrow \mathsf{In1}$

Inequality Termination (type 2)

ports: In = value inequality, 2 value derivatives, $\mathsf{Sup} = \emptyset$, Out = value inequality
example: "There's more water on the right hand side, ... it goes down there and up at the left, so it becomes leveled"
$(V_l < V_r,\ \delta V_l > 0,\ \delta V_r < 0 \,\&\, \text{—} \rightarrow V_l = V_r)$
forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In1} = [A_1 =/> A_2] \,\&\, \mathsf{In2} = [\delta A_1 < 0] \,\&\, \mathsf{In3} = [\delta A_2 > 0]$
THEN $\mathsf{Out} = [A_1 </= A_2]$
IF $\mathsf{In1} = [A_1 =/> A_2] \,\&\, \mathsf{In2} = [\delta A_1 = 0] \,\&\, \mathsf{In3} = [\delta A_2 > 0]$
THEN $\mathsf{Out} = [A_1 </= A_2]$
IF $\mathsf{In1} = [A_1 =/> A_2] \,\&\, \mathsf{In2} = [\delta A_1 < 0] \,\&\, \mathsf{In3} = [\delta A_2 = 0]$
THEN $\mathsf{Out} = [A_1 </= A_2]$

IF $\mathsf{In1} = [A_1 </= A_2] \,\&\, \mathsf{In2} = [\delta A_1 > 0] \,\&\, \mathsf{In3} = [\delta A_2 < 0]$
THEN $\mathsf{Out} = [A_1 =/> A_2]$

IF $\mathsf{In1} = [A_1 </= A_2] \,\&\, \mathsf{In2} = [\delta A_1 = 0] \,\&\, \mathsf{In3} = [\delta A_2 < 0]$
THEN $\mathsf{Out} = [A_1 =/> A_2]$
IF $\mathsf{In1} = [A_1 </= A_2] \,\&\, \mathsf{In2} = [\delta A_1 > 0] \,\&\, \mathsf{In3} = [\delta A_2 = 0]$
THEN $\mathsf{Out} = [A_1 =/> A_2]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{In2} \,\&\, \mathsf{In3} \rightarrow \mathsf{In1}$

Value Determination (type 1)

ports: In = value inequality, Sup = value definition, Out = quantity value
example: "The volumes are equal, therefore the balance will be leveled"
$(V_l = V_r \,\&\, \mathrm{Pos} = V_l - V_r \rightarrow \mathrm{Pos} = 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [A </=/> B] \,\&\, \mathsf{Sup} = [C = A - B]$ THEN $\mathsf{Out} = [C </=/> 0]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$
$\mathsf{In} \,\&\, \mathsf{Out} \rightarrow \mathsf{Sup}$

Value Determination (type 2)

ports: In = quantity values, Sup = value definition, Out = quantity value
example: "The mass of the container plus the mass of the liquid makes up a positive total mass"
$(M_c > 0,\ M_l > 0 \,\&\, M_t = M_c + M_l \rightarrow M_t > 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In1} = [A < 0] \,\&\, \mathsf{In2} = [B >/= 0] \,\&\, \mathsf{Sup} = [C = A - B]$ THEN $\mathsf{Out} = [C < 0]$
IF $\mathsf{In1} = [A </= 0] \,\&\, \mathsf{In2} = [B > 0] \,\&\, \mathsf{Sup} = [C = A - B]$ THEN $\mathsf{Out} = [C < 0]$
IF $\mathsf{In1} = [A = 0] \,\&\, \mathsf{In2} = [B = 0] \,\&\, \mathsf{Sup} = [C = A - B]$ THEN $\mathsf{Out} = [C = 0]$

IF $\mathsf{In1} = [A < 0] \,\&\, \mathsf{In2} = [B </= 0] \,\&\, \mathsf{Sup} = [C = A + B]$ THEN $\mathsf{Out} = [C < 0]$
IF $\mathsf{In1} = [A </= 0] \,\&\, \mathsf{In2} = [B < 0] \,\&\, \mathsf{Sup} = [C = A + B]$ THEN $\mathsf{Out} = [C < 0]$
IF $\mathsf{In1} = [A = 0] \,\&\, \mathsf{In2} = [B = 0] \,\&\, \mathsf{Sup} = [C = A + B]$ THEN $\mathsf{Out} = [C = 0]$

IF $\mathsf{In1} = [A > 0] \,\&\, \mathsf{In2} = [B >/= 0] \,\&\, \mathsf{Sup} = [C = A + B]$ THEN $\mathsf{Out} = [C > 0]$
IF $\mathsf{In1} = [A >/= 0] \,\&\, \mathsf{In2} = [B > 0] \,\&\, \mathsf{Sup} = [C = A + B]$ THEN $\mathsf{Out} = [C > 0]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{In1} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In2}$ (partially)
$\mathsf{Out} \,\&\, \mathsf{In2} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In1}$ (partially)
$\mathsf{Out} \,\&\, \mathsf{In1} \,\&\, \mathsf{In2} \rightarrow \mathsf{Sup}$ (partially)

Derivative Determination (type 1)

ports: In = value inequality, Sup = derivative definition, Out = quantity derivative
example: "The flow rate is larger at the right, therefore the balance will move up there"
$(F_l < Fl_r \,\&\, \delta\mathrm{Pos} = Fl_l - Fl_r \rightarrow \delta\mathrm{Pos} < 0)$

forward behaviour rules: $\mathsf{In} \,\&\, \mathsf{Sup} \rightarrow \mathsf{Out}$
IF $\mathsf{In} = [A </=/> B] \,\&\, \mathsf{Sup} = [\delta C = A - B]$ THEN $\mathsf{Out} = [\delta C </=/> 0]$

backward behaviour rules: $\mathsf{Out} \,\&\, \mathsf{Sup} \rightarrow \mathsf{In}$
$\mathsf{In} \,\&\, \mathsf{Out} \rightarrow \mathsf{Sup}$

Derivative Determination (type 2)

parts: In = quantity values, Sup = derivative definition, Out = quantity derivative

example: "Only the left flows, so the balance goes up [there]"
$(Fl_l > 0,\ Fl_r = 0 \,\&\, \delta\text{Pos} = Fl_l - Fl_r \rightarrow \delta\text{Pos} > 0)$

forward behaviour rules: In & Sup → Out
analogous to type 2 value determinations

backward behaviour rules: Out & In1 & Sup → In2 (partially)
Out & In2 & Sup → In1 (partially)
Out & In1 & In2 → Sup (partially)
analogous to type 2 value determinations

Appendix B. Algorithms: Hierarchical aggregation and diagnosis

Algorithm B.1 first collects all sets of fully-corresponding quantities (step (1)). All references to a quantity that belongs to such a set are replaced by a reference to a new quantity representing the whole set (step (2)). Now, all sets of points with identical expressions are merged to one point (step (3)), thereby 'flattening' network parts such as the one shown in Fig. 9. In steps (4) and (5), identical components between points, as well as components that have the same point as input and output, are removed.[15] The latter category of circular components consists of correspondences and proportionalities: after combining two quantities A and B, a correspondence component from $A > 0$ to $B > 0$ becomes one from $[A, B] > 0$ to $[A, B] > 0$.

Algorithm B.1 (*Hiding fully-corresponding quantities*).

(1) Find in the model all maximal sets of quantities $S = \{Q_1, \ldots, Q_n\}$ for which the following condition holds:

$$\forall Q_i \in S\ \exists Q_j \in S\colon\ (\mathit{corr}(Q_i, Q_j) \vee \mathit{corr}(Q_j, Q_i)) \wedge \mathit{pos\_prop}(Q_i, Q_j) \wedge \mathit{pos\_prop}(Q_j, Q_i).$$

(2) For each set $S = \{Q_1, \ldots, Q_n\}$ found, do the following:
   (a) find the set of all expressions that refer to a member Q_i of S;
   (b) in each of these expressions, replace each member of S with a new quantity $[Q_1, \ldots, Q_n]$.

(3) Within each state, merge all points that have equal expressions associated to them by removing all but one and reconnect all incoming and outgoing connections to this one remaining point.

(4) For each non-unique component C, let $C_1, \ldots, C_n$ be the set of components such that $\forall C_i, 1 \leqslant i \leqslant n$: $(\text{type}(C) = \text{type}(C_i),\ \text{inputs}(C) = \text{inputs}(C_i),\ \text{generic}(C) =$

[15] Removal of components refers to removal at the newly created level: all aggregation algorithms operate by first copying the complete model from the previous level, and then transforming the copy into the new model.

generic(C_i), and output(C) = output(C_i)) ∨ (inputs(C_i) = output(C_i) = in/output (C)). Add a decomposition[16] $C \to C, C_1, \ldots, C_n$.
(5) Remove each set of components $C_1, \ldots, C_n$ from the model.

Algorithm B.2 collects all submissive and continuity components from the copied base model (step (1)), and then removes them (step (2)). The input expression of the removed submissive component may not be used for anything else, in which case it can be removed as well, to avoid redundancy in the model. Therefore, the second step recursively checks for redundant components and points (recall that input(C) is a point). Step (3) adds the decomposition relation for transitions.

Algorithm B.2 (*Hiding submissive components*).
(1) Find the set S of all submissive and continuity components.
(2) Until S is empty, do the following for a component $C \in S$:
  (a) if input(C) is not input to any other component, then for all $C', C' \neq C$: output(C') = input(C), add C' to S;
  (b) remove component C from the model;
  (c) remove C from S.
(3) For each set $T \subset S$ of submissive termination and continuity components removed in a particular transition t, add a decomposition $S' \to S' \cup T$, where $S' = S \backslash T$ is the set of remaining termination components in t.
(4) For each set of removed components $C_1, \ldots, C_k$ with $\forall C_i \exists C_j, 1 \leqslant i < j \leqslant k$: output($C_i$) = input($C_j$), do the following:
  (a) if $C_1 \in S$: add a decomposition $C_n, \ldots, C_m \to C_n, \ldots, C_m, C_1, \ldots, C_k$, where $C_n, \ldots, C_m$ is the set of components with the same input as C_1 ('head' decomposition);
  (b) if $C_k \in S$: add a decomposition $C_n, \ldots, C_m \to C_n, \ldots, C_m, C_1, \ldots, C_k$, where $C_n, \ldots, C_m$ is the set of components with the same output as C_k ('tail' decomposition).

Algorithm B.3 first locates chains of connected components of the right type (step (1)). An extra condition is that no branching is allowed in a chain, except for branches to termination components (step 1(c)), i.e., chains should be straight sequences, and maximally separated from the rest of the model. Steps (2) to (4) make sure that if such a termination cannot be removed because not all of its inputs are 'removable', the chain will split at that point.

Algorithm B.3 (*Chunking transitive components*).
(1) Find in the model all maximal lists of components $L = [C_1, \ldots, C_n],\ n \geqslant 2$, for which the following conditions hold:
  (a) each C_i has the same type $t \in$ {quantity correspondence, quantity proportionality, inequality correspondence, inequality proportionality};

[16] The term 'decomposition' is used instead of 'aggregation' because the aggregated models are used for decomposition.

Table B.1
Possible predecessor chunking combinations

Component C_1		Key component C_2
(transitive) quantity correspondence	+	quantity influence
(transitive) inequality correspondence	+	inequality influence
(transitive) inequality correspondence	+	value determination (type 1)
(transitive) inequality correspondence	+	derivative determination (type 1)
(transitive) inequality proportionality	+	value determination (type 1)
(transitive) inequality proportionality	+	derivative determination (type 1)

(b) L is a chain:
$\forall C_i, \quad C_{i+1} \in L$: output$(C_i) =$ input(C_{i+1});
(c) L obeys the non-branching condition: $\nexists$ point P: $((P =$ output$(C_i),\ 1 \leqslant i < n)$
$\wedge (\exists C$: $P =$ input(C), $C \notin L \wedge$ type$(C) \neq$ termination$))$.

(2) Mark the input port of each transition component connected to a point in between C_1 and C_n for some L as 'removable'.
(3) Remove all transition components of which each input port is marked as 'removable'.
(4) For each $L = [C_1, \ldots, C_n]$, split the list in sub-lists $L_s = [C_i, \ldots, C_j], 1 \leqslant i < j \leqslant n$, such that $\nexists$ point P: $((P =$ output$(C_k),\ i \leqslant k < j) \wedge (P =$ input$(C),\ C \notin L_s))$.
(5) For each remaining sub-list $L_s = [C_i, \ldots, C_j]$ with $j > i$, do the following:
(a) create a new component C of type transitive $\langle \mathtt{t} \rangle$, where $\langle \mathtt{t} \rangle$ is the type of the members of L_s;
(b) generate behaviour rules relating input(C_i) to output(C_j);
(c) define input$(C) =$ input(C_i), output$(C) =$ output(C_j);
(d) add a decomposition $C \rightarrow C_i, \ldots, C_j$;
(e) remove all components in L_s, plus all points in between, from the model.

Algorithm B.4 combines pairs as mentioned in Table B.1 (step (1)). Important to note is that step 1(c) forbids branching to *other* types of components, but not to the ones mentioned in the right column of Table B.1. This way, it is possible to chunk two pairs C_1, C_2 and C_1, C_3 into two different combined types.

Algorithm B.4 (*Predecessor chunking*).
(1) Find in the model all pairs of components C_1, C_2 for which the following conditions hold:
(a) output$(C_1) =$ input(C_2);
(b) type(C_1) and type(C_2) appear as a row in Table B.1;
(c) output(C_1) is not input to any component of a type other than mentioned in the right column of Table B.1, or a termination type.
(2) For each pair C_1, C_2, do the following:
(a) create a new component C of type combined $\langle \mathtt{t} \rangle$ 1, where $\langle \mathtt{t} \rangle$ is the type of C_2;
(b) generate behaviour rules relating input(C_1) to output(C_2);
(c) define input$(C) =$ input(C_1), output$(C) =$ output(C_2);

(d) add a decomposition $C \rightarrow C_1, C_2$;
(e) remove all terminations T that have output(C_1) as an input, and for which output(T) is not input to any component.
(3) Remove all component pairs C_1, C_2 from the model.
(4) Remove all inactive paths from the model.

Algorithm B.5 is analogous to Algorithm B.4.

Algorithm B.5 (*Successor chunking*).
(1) Find in the model all maximal lists of components $L = [C_1, \ldots, C_n],\ n \geqslant 2$, for which the following conditions hold:
(a) C_1 is a key component or combined key component, i.e., it is of type (combined) influence, (combined) value determination, or (combined) derivative determination;
(b) L is a chain:
$\forall C_i, C_{i+1} \in L$: output($C_i$) = input($C_{i+1}$);
(c) L obeys the non-branching condition:
$\nexists$ point P: ((P = output(C_i), $1 \leqslant i < n$) $\wedge$ ($\exists C$: P = input(C), $C \notin L \wedge$ type(C) $\neq$ transition)).
(2) For each list L, remove all terminations T that have output(C_i), $1 \leqslant i < n$, as an input, and for which output(T) is not input to any component.
(3) For each list L do the following:
(a) create a new component C of type combined ⟨t⟩ 2, where ⟨t⟩ is the type of C_1;
(b) generate behaviour rules relating input(C_1) to output(C_n);
(c) define input(C) = input(C_1), output(C) = output(C_n);
(d) add a decomposition $C \rightarrow C_1, \ldots, C_n$;
(e) Remove L from the model.
(4) Remove all inactive paths from the model.

Algorithm B.6 not only refers to inputs and outputs of components, but also of *sets*. For a set S, input(S) and output(S) are defined as follows:

$$P \in \text{input}(S) \text{ iff } \exists C_i \in S\colon\ P = \text{input}(C_i) \wedge \nexists C_o \in S\colon\ P = \text{output}(C_o),$$
$$P \in \text{output}(S) \text{ iff } \nexists C_i \in S\colon\ P = \text{input}(C_i) \wedge \exists C_o \in S\colon\ P = \text{output}(C_o).$$

The algorithm first splits the set of all specification components according to the state they belong to (step (1)), and creates a new component for each state specification (step (2)). Then the termination components between a subsequent pair of states are combined into a new state transition component (steps (4), (5)). New points and connections are added by combining all connections that existed between the different sets (steps (7), (8)). The points that are to be generated in steps (7) and (8) may already exist in the case that a state has more than one successor or predecessor state.

Algorithm B.6 (*Grouping*).
(1) Find all sets S_i of specification components that belong to the same state i.
(2) For each S_i, create a new component C_i of type state specification.

(3) Add a decomposition $C_i \rightarrow S_i$.
(4) For each two sets S_i, S_j for which $\mathsf{state}(j) = \mathsf{succ}(\mathsf{state}(i))$, find the set T_{ij} of all termination components C for which $\mathsf{input}(C) \in \mathsf{input}(S_i)$ and $\mathsf{output}(C) \in \mathsf{output}(S_j)$.
(5) For each set of termination components T_{ij}, create a new component C_{ij} of type state transition.
(6) Add a decomposition $C_{ij} \rightarrow T_{ij}$.
(7) For each pair of components C_i, C_{ij}, if no point P_i already exists for which $\mathsf{output}(C_i) = P_i$, then create it, and define $\mathsf{output}(C_i) = P_i = \mathsf{input}(C_{ij})$.
(8) For each pair of components C_{ij}, C_j, if no point P_j already exists for which $\mathsf{output}(C_j) = P_j$, then create it, and define $\mathsf{output}(C_{ij}) = P_j = \mathsf{input}(C_j)$.
(9) For each point P_i (P_j) created in steps (6), (7), create expressions E_i (E_j) where

$$E_i = \bigcup \mathsf{expression}(P_k)\colon\ P_k \in (\mathsf{output}(S_i) \cup \mathsf{input}(T_{ij})) \quad \text{and}$$

$$E_j = \bigcup \mathsf{expression}(P_k)\colon\ P_k \in (\mathsf{output}(T_{ij}) \cup \mathsf{input}(S_j)).$$

(10) Remove all components of type other than state specification or state transition from the model.

Algorithm B.7 implements the basic GDE engine for computing candidates.

Algorithm B.7 (*Computing candidates*).

Conflict Recognition

Let *COMP* be the set of components in the model.
Let *OBS* be the set of new observations.

(1) Retrieve the label CfS_1 of the nogood node in the ATMS, constituting the current *conflict set*.
(2) Add each observation in *OBS* as a premise to the ATMS.
(3) For each possible subset of *COMP* that is not a superset of a nogood, call the prediction engine to derive new facts. Each new derivation is passed as a justification to the ATMS.
(4) Retrieve the new label CfS_2 of the nogood node in the ATMS.
(5) Determine the set of new conflicts $CfS_n = CfS_2 \backslash CfS_1$. If CfS_n is empty, then stop.

Candidate Generation

Update the candidate set for each new conflict by *incremental set covering*:

(1) For each candidate that does not cover any of the new conflicts, replace it with a set of new candidates, each of which contains the old candidate plus one element from the new conflict;
(2) Minimalise the resulting set by removing each candidate that is subsumed or duplicated by another candidate.

If not already at the lowest level (step (1)), Algorithm B.8 select the component with the maximum unnormalised probability as defined in the candidate discrimination algorithm (steps (2) and (3)), and decompose this component into its lower level representative. This

is done by finding the decomposition relation that contains the selected component in the antecedent (step (4)).

Algorithm B.8 (*Decomposition*).

Let the input be a candidate set *CaS* at level l.

(1) If $l = 0$, then stop.
(2) If $l > 0$, calculate the unnormalised probability of each component $Cp \in Ca_i$ for candidates $Ca_i \in CaS$.
(3) Select the component $C_{\max}$ with the maximum unnormalised probability. If more than one component has the maximum value, pick one randomly.
(4) Find the decomposition relation $M_l \rightarrow M_{l-1}$ with $C_{\max} \in M_l$.
(5) Transfer all known observations to the new model M_{l-1}.
(6) Call the main diagnostic Algorithm B.7 on M_{l-1}.
(7) If the diagnostic algorithm returns an empty set of candidates, return to step (3) and select another component.

References

[1] J.R. Anderson, Rules of the Mind, Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.
[2] J.R. Anderson, C.F. Boyle, A.T. Corbett, M. Lewis, Cognitive modeling and intelligent tutoring, Artificial Intelligence 42 (1990) 7–49.
[3] A. Barr, M. Beard, R.C. Atkinson, The computer as a tutorial laboratory: The Stanford BIP project, Internat. J. Man-Machine Studies 8 (1976) 567–596.
[4] S. Beller, H.U. Hoppe, Deductive error reconstruction and classification in a logic programming framework, in: P. Brna, S. Ohlsson, H. Pain (Eds.), Proc. World Conference on Artificial Intelligence in Education, Charlottesville, VA, AACE, 1993, pp. 433–440.
[5] V.R. Benjamins, Problem solving methods for diagnosis, Ph.D. Thesis, University of Amsterdam, 1993.
[6] J.G. Bonar, E. Soloway, Preprogramming knowledge: A major source of misconceptions in novice programmers, Human Computer Interaction 1 (2) (1985) 133–161.
[7] B. Bredeweg, Introducing meta-levels to qualitative reasoning, Appl. Artificial Intelligence 3 (2) (1989) 85–100.
[8] B. Bredeweg, Expertise in qualitative prediction of behaviour, Ph.D. Thesis, University of Amsterdam, 1992.
[9] B. Bredeweg, J.A. Breuker, 'Device Models' for model-based diagnosis of student behaviour, in: P. Brna, S. Ohlsson, H. Pain (Eds.), Proc. World Conference on Artificial Intelligence in Education, Charlottesville, VA, AACE, 1993, pp. 441–448.
[10] B. Bredeweg, K. de Koning, C. Schut, Modelling the influence of non-changing quantities, in: J. Wainer, A. Carvalho (Eds.), Advances in Artificial Intelligence, Springer, Berlin, 1995, pp. 131–140.
[11] B. Bredeweg, C. Schut, Cognitive plausibility of a conceptual framework for modeling problem solving expertise, in: Proc. Cognitive Science Society Conference, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991, pp. 473–479.
[12] B. Bredeweg, R.G.F. Winkels, Qualitative models in interactive learning environments: An introduction, Interactive Learning Environments 5 (1) (1998) 1–18.
[13] J.A. Breuker (Ed.), EUROHELP: Developing Intelligent Help Systems, EC, Amsterdam, 1990.
[14] J.A. Breuker, Components of problem solving, in: L. Steels, A.Th. Schreiber, W. van de Velde (Eds.), A Future for Knowledge Acquisition: Proceedings of the European Knowledge Acquisition Workshop-94, Springer, Berlin, 1994, pp. 118–136.
[15] J.A. Breuker, B.J. Wielinga, Model-driven knowledge acquisition, in: P. Guida, G. Tasso (Eds.), Topics in the Design of Expert Systems, North-Holland, Amsterdam, 1989, pp. 265–296.
[16] J.S. Brown, Uses of AI and advanced computer technology in education, in: R.J. Seidel, M. Rubin (Eds.), Computers and Communications: Implications for Education, Academic Press, New York, 1977.

[17] J.S. Brown, R.R. Burton, Diagnostic models for procedural bugs in basic mathematical skills, Cognitive Sci. 2 (1978) 155–192.
[18] J.S. Brown, K. van Lehn, Repair theory: A generative theory of bugs in procedural skills, Cognitive Sci. 4 (1980) 379–426.
[19] R.R. Burton, Diagnosing bugs in a simple procedural skill, in: D.H. Sleeman, J.S. Brown (Eds.), Intelligent Tutoring Systems, Academic Press, London, 1982.
[20] T. Bylander, D. Allemang, M.C. Tanner, J.R. Josephson, The computational complexity of abduction, Artificial Intelligence 49 (1991) 25–60.
[21] J.R. Carbonell, AI in CAI: An artificial intelligence approach to computer-assisted instruction, IEEE Trans. Man-Machine Systems 11 (4) (1970) 190–202.
[22] B. Carr, I.P. Goldstein, Overlays: A theory of modeling for computer aided instruction, AI Memo 406, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1977.
[23] W.J. Clancey, The epistemology of a rule based system—A framework for explanation, Artificial Intelligence 20 (1983) 215–251.
[24] W.J. Clancey, Qualitative student models, in: J.F. Traub (Ed.), Annual Review of Computer Science, Vol. 1, Annual Review Inc., Palo Alto, CA, 1986, pp. 381–450.
[25] A. Collins, A.L. Stevens, Goals and strategies for inquiry teachers, in: R. Glaser (Ed.), Advances in Instructional Psychology, Vol. II, Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.
[26] T. de Jong, L. Sarti (Eds.), Design and Production of Multimedia and Simulation-Based Learning Material, Kluwer, Dordrecht, the Netherlands, 1994.
[27] J. de Kleer, Focusing on probable diagnoses, in: Proc. AAAI-91, Anaheim, CA, 1991, pp. 842–848.
[28] J. de Kleer, J.S. Brown, A qualitative physics based on confluences, Artificial Intelligence 24 (1984) 7–83.
[29] J. de Kleer, J.S. Brown, Model-based diagnosis in SOPHIE III, in: W.C. Hamscher, L. Console, J. de Kleer (Eds.), Readings in Model-Based Diagnosis, Morgan Kaufmann, San Mateo, CA, 1992, pp. 179–205.
[30] J. de Kleer, B.C. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1987) 97–130.
[31] K. de Koning, Model-Based Reasoning about Learner Behaviour, IOS Press, Amsterdam, 1997.
[32] K. de Koning, B. Bredeweg, A framework for teaching qualitative models, in: A. Cohn (Ed.), Proc. 11th European Conference on Artificial Intelligence, Wiley, New York, 1994, pp. 197–202.
[33] K. de Koning, B. Bredeweg, Qualitative reasoning in tutoring interactions, J. Interactive Learning Environments 5 (1998) 65–80.
[34] K. de Koning, B. Bredeweg, C. Schut, J.A. Breuker, Dynamic model progression, in: Proc. East–West Conference on Computer Technologies in Education, Moscow, International Center for Scientific and Technical Information, 1994, pp. 136–141.
[35] P. Dillenbourg, J.A. Self, A framework for learner modelling, Interactive Learning Environments 2 (1992) 111–137.
[36] M. Elsom-Cook (Ed.), Guided Discovery Tutoring: A Framework for ICAI Research, Paul Chapman, London, 1990.
[37] B.C. Falkenhainer, K.D. Forbus, Compositional modeling: Finding the right model for the job, Artificial Intelligence 51 (1991) 95–143.
[38] K.D. Forbus, Qualitative process theory, Artificial Intelligence 24 (1984) 85–168.
[39] K.D. Forbus, The qualitative process engine, in: D.S. Weld, J. de Kleer (Eds.), Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Mateo, CA, 1990, pp. 220–235.
[40] K.D. Forbus, Towards tutor compilers: Self-explanatory simulations as an enabling technology, in: L. Birnbaum (Ed.), Proc. 3rd Internat. Conference on the Learning Sciences, Evanston, IL, 1991.
[41] K.D. Forbus, Using qualitative physics to create articulate educational software, IEEE Expert 12 (3) (1997) 32–41.
[42] P.O. Gautier, T.R. Gruber, Generating explanations of device behavior using compositional modeling causal ordering, in: Proc. AAAI-96, Portland, OR, MIT Press, Cambridge, MA, 1996, pp. 264–270.
[43] M.R. Genesereth, The use of design descriptions in automated diagnosis, Artificial Intelligence 24 (1984) 411–436.
[44] D.R. Gentner, Toward an intelligent computer tutor, in: H. O'Neil (Ed.), Procedures for Instructional Systems Development, Academic Press, New York, 1979.
[45] J.E. Greer, G.I. McCalla (Eds.), Student Modelling: The Key to Individualized Knowledge-Based Instruction, Series F: Computer and System Sciences, Vol. 125, Springer, Berlin, 1994.

[46] W.C. Hamscher, L. Console, J. de Kleer, Introduction to Chapter 7: Hierarchies, in: Readings in Model-Based Diagnosis, Morgan Kaufmann, San Mateo, CA, 1992.
[47] W.C. Hamscher, L. Console, J. de Kleer (Eds.), Readings in Model-Based Diagnosis, Morgan Kaufmann, San Mateo, CA, 1992.
[48] H.U. Hoppe, Deductive error diagnosis and inductive error generalization for intelligent tutoring systems, J. Artificial Intelligence in Education 5 (1) (1994) 27–49.
[49] X. Huang, G.I. McCalla, E. Neufeld, Using attention in belief revision, in: Proc. AAAI-91, Anaheim, CA, 1991, pp. 275–280.
[50] W.L. Johnson, E. Soloway, PROUST: An automatic debugger for pascal programs, in: G. Kearsley (Ed.), Artificial Intelligence and Instruction: Applications and Methods, Addison-Wesley, Reading, MA, 1987, pp. 49–67.
[51] P. Kamsteeg, Teaching problem solving by computer, Ph.D. Thesis, University of Amsterdam, 1994.
[52] J.E. Laird, A. Newell, P.S. Rosenbloom, SOAR: An architecture for general intelligence, Artificial Intelligence 33 (1987) 1–64.
[53] J.E. Laird, P.S. Rosenbloom, A. Newell, Chunking in SOAR: The anatomy of a general learning mechanism, Machine Learning 1 (1986) 11–46.
[54] A.Y. Levy, Y. Iwasaki, H. Motoda, Relevance reasoning to guide compositional modelling, in: Proc. 6th International Workshop on Qualitative Reasoning about Physical Systems, Edinburgh, Scotland, Heriot-Watt University, 1992, pp. 7–21.
[55] R.S. Mallory, B.W. Porter, B.J. Kuipers, Comprehending complex behavior graphs through abstraction, in: Y. Iwasaki, A. Farquhar (Eds.), Proc. 10th International Workshop on Qualitative Reasoning, AAAI Press, Menlo Park, CA, 1996, pp. 137–146. AAAI Technical Report WS-96-01.
[56] T.M. Mitchell, Generalization as search, Artificial Intelligence 18 (1982) 203–226.
[57] I. Mozetič, Hierarchical model-based diagnosis, Internat. J. Man-Machine Studies 35 (3) (1991) 329–362.
[58] S. Ohlsson, Some principles of intelligent tutoring, Instructional Science 14 (1986) 293–326.
[59] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas, Basic Books, New York, 1980.
[60] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1987) 57–96.
[61] C. Schut, B. Bredeweg, Automatic enhancement of model parsimony, in: D.S. Weld (Ed.), Proc. 7th International Workshop on Qualitative Reasoning about Physical Systems, Seattle, WA, University of Washington, 1993, pp. 194–203.
[62] J.A. Self, Model-based cognitive diagnosis, User Modeling and User-Adapted Interaction 3 (1993) 86–106.
[63] J.A. Self, Formal approaches to student modelling, in: G.I. McCalla, J. Greer (Eds.), Student Modelling: The Key to Individualized Knowledge-Based Instruction, Springer, Berlin, 1994, pp. 295–352.
[64] J.A. Sime, R.R. Leitch, A learning environment based on multiple qualitative models, in: G. Frasson, G. Gauthier, G.I. McCalla (Eds.), Proc. 2nd International ITS Conference, Edinburgh, Scotland, UK, Lecture Notes in Computer Science, Vol. 608, Springer, Berlin, 1992, pp. 116–123.
[65] D.H. Sleeman, Inferring student models for intelligent computer-aided instruction, in: R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, Palo Alto, CA, 1983, pp. 483–510.
[66] D. Subramanian, M.R. Genesereth, The relevance of irrelevance, in: Proc. IJCAI-87, Milan, Italy, 1987, pp. 416–422.
[67] M.B. Twidale, Coping with the variety of student actions in simulations, in: Proc. EARLI Conference, Aix en Provence, 1993.
[68] W. van de Velde, An overview of CommonKADS, in: J.A. Breuker, W. van de Velde (Eds.), The CommonKADS Library for Expertise Modelling, Chapter 2, IOS Press, Amsterdam, 1994, pp. 9–30.
[69] A. van der Hulst, Cognitive tools, Ph.D. Thesis, University of Amsterdam, 1996.
[70] F. van Harmelen, Meta-level Inference Systems, Research Notes in AI, Pitmann/Morgan Kaufmann, London/San Mateo, CA, 1991.
[71] M. van Someren, Y.F. Barnard, J.A.C. Sandberg, The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes, Academic Press, London, 1994.
[72] K. van Lehn, Learning one subprocedure per lesson, Artificial Intelligence 31 (1) (1987) 1–40.
[73] D.S. Weld, Approximation reformulations, in: Proc. AAAI-90, Boston, MA, AAAI Press/MIT Press, Menlo Park, CA, 1990, pp. 407–412.

[74] D.S. Weld, S. Addanki, Task-driven model abstraction, in: B. Faltings, P. Struss (Eds.), Recent Advances in Qualitative Physics, MIT Press, Cambridge, MA, 1992.
[75] E. Wenger, Artificial Intelligence and Tutoring Systems, Morgan Kaufmann, Los Altos, CA, 1987.
[76] B.Y. White, J.R. Frederiksen, Causal model progressions as a foundation for intelligent learning environments, Artificial Intelligence 42 (1990) 99–157.
[77] J. Wielemaker, SWI-Prolog 2.8: Reference Manual, Social Science Informatics (SWI), University of Amsterdam, 1997. http://www.swi.psy.uva.nl/usr/jan/SWI-Prolog/Manual/Title.html.
[78] B.J. Wielinga, J.A. Breuker, Models of expertise, in: Proc. ECAI-86, Brighton, UK, 1986, pp. 306–318.
[79] B.J. Wielinga, A.Th. Schreiber, J.A. Breuker, KADS: A modelling approach to knowledge engineering, Knowledge Acquisition J. 4 (1) (1992) 5–53.
[80] R.G.F. Winkels, Explorations in Intelligent Tutoring and Help, IOS Press, Amsterdam, 1992.